



## Formalization of Bachmair and Ganzinger's Ordered Resolution Prover

Schlichtkrull, Anders; Blanchette, Jasmin Christian; Traytel, Dmitriy; Waldmann, Uwe

*Published in:*  
Archive of Formal Proofs

*Publication date:*  
2018

*Document Version*  
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

*Citation (APA):*  
Schlichtkrull, A., Blanchette, J. C., Traytel, D., & Waldmann, U. (2018). Formalization of Bachmair and Ganzinger's Ordered Resolution Prover. *Archive of Formal Proofs*, 1-117.

---

### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# Formalization of Bachmair and Ganzinger’s Ordered Resolution Prover

Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann

January 22, 2018

## Abstract

This Isabelle/HOL formalization covers Sections 2 to 4 of Bachmair and Ganzinger’s “Resolution Theorem Proving” chapter in the *Handbook of Automated Reasoning*. This includes soundness and completeness of unordered and ordered variants of ground resolution with and without literal selection, the standard redundancy criterion, a general framework for refutational theorem proving, and soundness and completeness of an abstract first-order prover.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Map Function on Two Parallel Lists</b>	<b>2</b>
<b>3</b>	<b>Liminf of Lazy Lists</b>	<b>4</b>
<b>4</b>	<b>Relational Chains over Lazy Lists</b>	<b>6</b>
4.1	Chains . . . . .	7
4.2	Full Chains . . . . .	16
<b>5</b>	<b>Clausal Logic</b>	<b>17</b>
5.1	Literals . . . . .	17
5.2	Clauses . . . . .	20
<b>6</b>	<b>Herbrand Intepretation</b>	<b>22</b>
<b>7</b>	<b>Abstract Substitutions</b>	<b>23</b>
7.1	Library . . . . .	24
7.2	Substitution Operators . . . . .	24
7.3	Substitution Lemmas . . . . .	27
7.3.1	Identity Substitution . . . . .	27
7.3.2	Associativity of Composition . . . . .	28
7.3.3	Compatibility of Substitution and Composition . . . . .	28
7.3.4	“Commutativity” of Membership and Substitution . . . . .	29
7.3.5	Signs and Substitutions . . . . .	29
7.3.6	Substitution on Literal(s) . . . . .	29
7.3.7	Substitution on Empty . . . . .	29
7.3.8	Substitution on a Union . . . . .	31
7.3.9	Substitution on a Singleton . . . . .	31
7.3.10	Substitution on <i>op</i> # . . . . .	31
7.3.11	Substitution on <i>tl</i> . . . . .	32
7.3.12	Substitution on <i>op</i> ! . . . . .	32
7.3.13	Substitution on Various Other Functions . . . . .	32
7.3.14	Renamings . . . . .	33
7.3.15	Monotonicity . . . . .	34
7.3.16	Size after Substitution . . . . .	34
7.3.17	Variable Disjointness . . . . .	34

7.3.18	Ground Expressions and Substitutions . . . . .	35
7.3.19	Subsumption . . . . .	37
7.3.20	Unifiers . . . . .	37
7.3.21	Most General Unifier . . . . .	37
7.3.22	Generalization and Subsumption . . . . .	37
7.4	Most General Unifiers . . . . .	40
<b>8</b>	<b>Refutational Inference Systems</b>	<b>41</b>
8.1	Preliminaries . . . . .	41
8.2	Refutational Completeness . . . . .	42
8.3	Compactness . . . . .	43
<b>9</b>	<b>Candidate Models for Ground Resolution</b>	<b>45</b>
<b>10</b>	<b>Ground Unordered Resolution Calculus</b>	<b>51</b>
10.1	Inference Rule . . . . .	51
10.2	Inference System . . . . .	53
<b>11</b>	<b>Ground Ordered Resolution Calculus with Selection</b>	<b>53</b>
11.1	Inference Rule . . . . .	53
11.2	Inference System . . . . .	60
<b>12</b>	<b>Theorem Proving Processes</b>	<b>60</b>
<b>13</b>	<b>The Standard Redundancy Criterion</b>	<b>65</b>
<b>14</b>	<b>First-Order Ordered Resolution Calculus with Selection</b>	<b>70</b>
14.1	Library . . . . .	71
14.2	First-Order Logic . . . . .	71
14.3	Calculus . . . . .	72
14.4	Soundness . . . . .	72
14.5	Other Basic Properties . . . . .	75
14.6	Inference System . . . . .	75
14.7	Lifting . . . . .	78
<b>15</b>	<b>An Ordered Resolution Prover for First-Order Clauses</b>	<b>90</b>

# 1 Introduction

Bachmair and Ganzinger’s “Resolution Theorem Proving” chapter in the *Handbook of Automated Reasoning* is the standard reference on the topic. It defines a general framework for propositional and first-order resolution-based theorem proving. Resolution forms the basis for superposition, the calculus implemented in many popular automatic theorem provers.

This Isabelle/HOL formalization covers Sections 2.1, 2.2, 2.4, 2.5, 3, 4.1, 4.2, and 4.3 of Bachmair and Ganzinger’s chapter. Section 2 focuses on preliminaries. Section 3 introduces unordered and ordered variants of ground resolution with and without literal selection and proves them refutationally complete. Section 4.1 presents a framework for theorem provers based on refutation and saturation. Finally, Section 4.2 generalizes the refutational completeness argument and introduces the standard redundancy criterion, which can be used in conjunction with ordered resolution. Section 4.3 lifts the result to a first-order prover, specified as a calculus. Figure 1 shows the corresponding Isabelle theory structure.

# 2 Map Function on Two Parallel Lists

```
theory Map2
  imports Main
begin
```

This theory defines a map function that applies a (curried) binary function elementwise to two parallel lists.

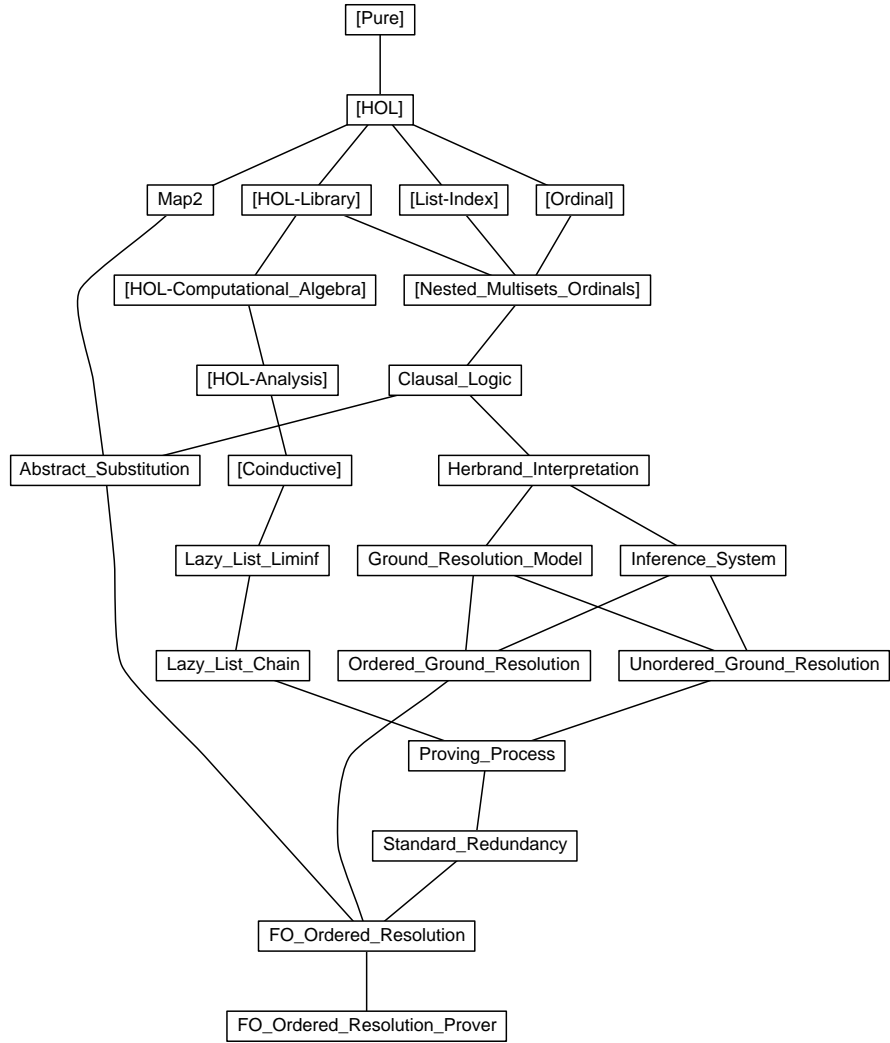


Figure 1: Theory dependency graph

The definition is taken from [https://www.isa-afp.org/browser\\_info/current/AFP/Jinja/Listn.html](https://www.isa-afp.org/browser_info/current/AFP/Jinja/Listn.html).

**abbreviation**  $\text{map2} :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list} \Rightarrow 'c \text{ list}$  **where**  
 $\text{map2 } f \text{ xs ys} \equiv \text{map } (\text{case\_prod } f) (\text{zip xs ys})$

**lemma**  $\text{map2\_empty\_iff}[simp]: \text{map2 } f \text{ xs ys} = [] \longleftrightarrow \text{xs} = [] \vee \text{ys} = []$   
**by** (metis Nil.is\_map\_conv list.exhaust list.simps(3) zip.simps(1) zip.Cons.Cons zip.Nil)

**lemma**  $\text{image\_map2}: \text{length } t = \text{length } s \Longrightarrow g \cdot \text{set } (\text{map2 } f \text{ t s}) = \text{set } (\text{map2 } (\lambda a \ b. g (f a b)) \text{ t s})$   
**by** auto

**lemma**  $\text{map2\_tl}: \text{length } t = \text{length } s \Longrightarrow \text{map2 } f (\text{tl } t) (\text{tl } s) = \text{tl } (\text{map2 } f \text{ t s})$   
**by** (metis (no\_types, lifting) hd.Cons\_tl list.sel(3) map2\_empty\_iff map\_tl tl\_Nil zip.Cons.Cons)

**lemma**  $\text{map\_zip\_assoc}$ :  
 $\text{map } f (\text{zip } (\text{zip xs ys}) \text{ zs}) = \text{map } (\lambda(x, y, z). f ((x, y), z)) (\text{zip xs } (\text{zip ys zs}))$   
**by** (induct zs arbitrary: xs ys) (auto simp add: zip.simps(2) split: list.splits)

**lemma**  $\text{set\_map2\_ex}$ :  
**assumes**  $\text{length } t = \text{length } s$   
**shows**  $\text{set } (\text{map2 } f \text{ s t}) = \{x. \exists i < \text{length } t. x = f (s ! i) (t ! i)\}$   
**proof** (rule; rule)  
**fix**  $x$   
**assume**  $x \in \text{set } (\text{map2 } f \text{ s t})$   
**then obtain**  $i$  **where**  $i\_p: i < \text{length } (\text{map2 } f \text{ s t}) \wedge x = \text{map2 } f \text{ s t} ! i$   
**by** (metis in\_set\_conv\_nth)  
**from**  $i\_p$  **have**  $i < \text{length } t$   
**by** auto  
**moreover from this**  $i\_p$  **have**  $x = f (s ! i) (t ! i)$   
**using**  $\text{assms}$  **by** auto  
**ultimately show**  $x \in \{x. \exists i < \text{length } t. x = f (s ! i) (t ! i)\}$   
**using**  $\text{assms}$  **by** auto  
**next**  
**fix**  $x$   
**assume**  $x \in \{x. \exists i < \text{length } t. x = f (s ! i) (t ! i)\}$   
**then obtain**  $i$  **where**  $i\_p: i < \text{length } t \wedge x = f (s ! i) (t ! i)$   
**by** auto  
**then have**  $i < \text{length } (\text{map2 } f \text{ s t})$   
**using**  $\text{assms}$  **by** auto  
**moreover from**  $i\_p$  **have**  $x = \text{map2 } f \text{ s t} ! i$   
**using**  $\text{assms}$  **by** auto  
**ultimately show**  $x \in \text{set } (\text{map2 } f \text{ s t})$   
**by** (metis in\_set\_conv\_nth)  
**qed**  
**end**

### 3 Liminf of Lazy Lists

**theory** *Lazy\_List\_Liminf*  
**imports** *Coinductive.Coinductive\_List*  
**begin**

Lazy lists, as defined in the *Archive of Formal Proofs*, provide finite and infinite lists in one type, defined coinductively. The present theory introduces the concept of the union of all elements of a lazy list of sets and the limit of such a lazy list. The definitions are stated more generally in terms of lattices. The basis for this theory is Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

**definition**  $\text{Sup\_llist} :: 'a \text{ set} \text{ llist} \Rightarrow 'a \text{ set}$  **where**  
 $\text{Sup\_llist } Xs = (\bigcup i \in \{i. \text{enat } i < \text{llength } Xs\}. \text{lnth } Xs \ i)$

**lemma**  $\text{lnth\_subset\_Sup\_llist}: \text{enat } i < \text{llength } xs \Longrightarrow \text{lnth } xs \ i \subseteq \text{Sup\_llist } xs$   
**unfolding**  $\text{Sup\_llist\_def}$  **by** auto

**lemma** *Sup\_llist\_LNil*[simp]: *Sup\_llist LNil = {}*  
**unfolding** *Sup\_llist\_def* **by** *auto*

**lemma** *Sup\_llist\_LCons*[simp]: *Sup\_llist (LCons X Xs) = X ∪ Sup\_llist Xs*  
**unfolding** *Sup\_llist\_def*  
**proof** (*intro subset\_antisym subsetI*)  
**fix** *x*  
**assume** *x ∈ (⋃ i ∈ {i. enat i < llength (LCons X Xs)}. lnth (LCons X Xs) i)*  
**then obtain** *i* **where** *len: enat i < llength (LCons X Xs)* **and** *nth: x ∈ lnth (LCons X Xs) i*  
**by** *blast*  
**from** *nth* **have** *x ∈ X ∨ i > 0 ∧ x ∈ lnth Xs (i - 1)*  
**by** (*metis lnth\_LCons' neq0\_conv*)  
**then have** *x ∈ X ∨ (∃ i. enat i < llength Xs ∧ x ∈ lnth Xs i)*  
**by** (*metis len Suc\_pred' eSuc.enat iless.Suc.eq less\_irrefl llength\_LCons not\_less order\_trans*)  
**then show** *x ∈ X ∪ (⋃ i ∈ {i. enat i < llength Xs}. lnth Xs i)*  
**by** *blast*  
**qed** ((*auto*)[], *metis i0\_lb lnth\_0 zero\_enat\_def, metis Suc\_ile\_eq lnth\_Suc\_LCons*)

**lemma** *lhd\_subset\_Sup\_llist*:  $\neg \text{lnull } Xs \implies \text{lhd } Xs \subseteq \text{Sup\_llist } Xs$   
**by** (*cases Xs*) *simp\_all*

**definition** *Sup\_upto\_llist* :: *'a set llist  $\Rightarrow$  nat  $\Rightarrow$  'a set* **where**  
*Sup\_upto\_llist Xs j = (⋃ i ∈ {i. enat i < llength Xs ∧ i ≤ j}. lnth Xs i)*

**lemma** *Sup\_upto\_llist\_mono*:  $j \leq k \implies \text{Sup\_upto\_llist } Xs\ j \subseteq \text{Sup\_upto\_llist } Xs\ k$   
**unfolding** *Sup\_upto\_llist\_def* **by** *auto*

**lemma** *Sup\_upto\_llist\_subset\_Sup\_llist*:  $j \leq k \implies \text{Sup\_upto\_llist } Xs\ j \subseteq \text{Sup\_llist } Xs$   
**unfolding** *Sup\_llist\_def Sup\_upto\_llist\_def* **by** *auto*

**lemma** *elem\_Sup\_llist\_imp\_Sup\_upto\_llist*:  $x \in \text{Sup\_llist } Xs \implies \exists j. x \in \text{Sup\_upto\_llist } Xs\ j$   
**unfolding** *Sup\_llist\_def Sup\_upto\_llist\_def* **by** *blast*

**lemma** *finite\_Sup\_llist\_imp\_Sup\_upto\_llist*:  
**assumes** *finite X* **and**  $X \subseteq \text{Sup\_llist } Xs$   
**shows**  $\exists k. X \subseteq \text{Sup\_upto\_llist } Xs\ k$   
**using** *assms*  
**proof** *induct*  
**case** (*insert x X*)  
**then have** *x: x ∈ Sup\_llist Xs* **and** *X: X ⊆ Sup\_llist Xs*  
**by** *simp+*  
**from** *x* **obtain** *k* **where** *k: x ∈ Sup\_upto\_llist Xs k*  
**using** *elem\_Sup\_llist\_imp\_Sup\_upto\_llist* **by** *fast*  
**from** *X* **obtain** *k'* **where** *k': X ⊆ Sup\_upto\_llist Xs k'*  
**using** *insert.hyps(3)* **by** *fast*  
**have** *insert x X ⊆ Sup\_upto\_llist Xs (max k k')*  
**using** *k k'*  
**by** (*metis insert\_absorb insert\_subset Sup\_upto\_llist\_mono max.cobounded2 max commute order.trans*)  
**then show** *?case*  
**by** *fast*  
**qed** *simp*

**definition** *Liminf\_llist* :: *'a set llist  $\Rightarrow$  'a set* **where**  
*Liminf\_llist Xs = (⋃ i ∈ {i. enat i < llength Xs}. ⋂ j ∈ {j. i ≤ j ∧ enat j < llength Xs}. lnth Xs j)*

**lemma** *Liminf\_llist\_subset\_Sup\_llist*:  $\text{Liminf\_llist } Xs \subseteq \text{Sup\_llist } Xs$   
**unfolding** *Liminf\_llist\_def Sup\_llist\_def* **by** *fast*

**lemma** *Liminf\_llist\_LNil*[simp]: *Liminf\_llist LNil = {}*  
**unfolding** *Liminf\_llist\_def* **by** *simp*

```

lemma Liminf_llist_LCons:
  Liminf_llist (LCons X Xs) = (if lnull Xs then X else Liminf_llist Xs) (is ?lhs = ?rhs)
proof (cases lnull Xs)
  case nnull: False
  show ?thesis
  proof
  {
    fix x
    assume  $\exists i. \text{enat } i \leq \text{llength } Xs$ 
     $\wedge (\forall j. i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
    then have  $\exists i. \text{enat } (Suc\ i) \leq \text{llength } Xs$ 
     $\wedge (\forall j. Suc\ i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
    by (cases llength Xs,
      metis not_lnull_conv[THEN iffD1, OF nnull] Suc.le_D eSuc.enat eSuc.ile_mono
      llength_LCons not_less_eq_eq zero_enat_def zero_le,
      metis Suc.leD enat_ord_code(3))
    then have  $\exists i. \text{enat } i < \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j)$ 
    by (metis Suc.ile_eq Suc.n_not_le_n lift_Suc_mono.le lnth_Suc_LCons nat.le_linear)
  }
  then show ?lhs  $\subseteq$  ?rhs
  by (simp add: Liminf_llist_def nnull) (rule subsetI, simp)

  {
    fix x
    assume  $\exists i. \text{enat } i < \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j)$ 
    then obtain i where
      i: enat i < llength Xs and
      j:  $\forall j. i \leq j \wedge \text{enat } j < \text{llength } Xs \longrightarrow x \in \text{lnth } Xs\ j$ 
    by blast

    have enat (Suc i) ≤ llength Xs
    using i by (simp add: Suc.ile_eq)
    moreover have  $\forall j. Suc\ i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j$ 
    using Suc.ile_eq Suc.le_D j by force
    ultimately have  $\exists i. \text{enat } i \leq \text{llength } Xs \wedge (\forall j. i \leq j \wedge \text{enat } j \leq \text{llength } Xs \longrightarrow x \in \text{lnth } (LCons X Xs) j)$ 
    by blast
  }
  then show ?rhs  $\subseteq$  ?lhs
  by (simp add: Liminf_llist_def nnull) (rule subsetI, simp)
qed
qed (simp add: Liminf_llist_def enat_0_iff(1))

lemma lfinite_Liminf_llist: lfinite Xs  $\implies$  Liminf_llist Xs = (if lnull Xs then {} else llast Xs)
proof (induction rule: lfinite_induct)
  case (LCons xs)
  then obtain y ys where
    xs: xs = LCons y ys
  by (meson not_lnull_conv)
  show ?case
  unfolding xs by (simp add: Liminf_llist_LCons LCons.IH[unfolded xs, simplified] llast_LCons)
qed (simp add: Liminf_llist_def)

lemma Liminf_llist_ltl:  $\neg \text{lnull } (\text{ltl } Xs) \implies \text{Liminf\_llist } Xs = \text{Liminf\_llist } (\text{ltl } Xs)$ 
by (metis Liminf_llist_LCons lhd_LCons_ltl lnull_ltlI)

```

end

## 4 Relational Chains over Lazy Lists

```

theory Lazy_List_Chain
imports HOL-Library.BNF_Corec Lazy_List_Liminf
begin

```

A chain is a lazy lists of elements such that all pairs of consecutive elements are related by a given relation. A full chain is either an infinite chain or a finite chain that cannot be extended. The inspiration for this theory is Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

## 4.1 Chains

**coinductive** *chain* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a llist ⇒ bool **for** *R* :: 'a ⇒ 'a ⇒ bool **where**  
*chain\_singleton*: *chain* *R* (LCons *x* LNil)  
| *chain\_cons*: *chain* *R* *xs* ⇒ *R* *x* (lhd *xs*) ⇒ *chain* *R* (LCons *x* *xs*)

**lemma**

*chain\_LNil[simp]*: ¬ *chain* *R* LNil **and**  
*chain\_not\_lnull*: *chain* *R* *xs* ⇒ ¬ *lnull* *xs*  
**by** (*auto elim*: *chain.cases*)

**lemma** *chain\_lappend*:

**assumes**  
*r\_xs*: *chain* *R* *xs* **and**  
*r\_ys*: *chain* *R* *ys* **and**  
*mid*: *R* (llast *xs*) (lhd *ys*)  
**shows** *chain* *R* (*lappend* *xs* *ys*)  
**proof** (*cases* *lfinite* *xs*)  
**case** *True*  
**then show** ?thesis  
**using** *r\_xs* *mid*  
**proof** (*induct* *rule*: *lfinite.induct*)  
**case** (*lfinite\_LConsI* *xs* *x*)  
**note** *fin* = *this*(1) **and** *ih* = *this*(2) **and** *r\_xxs* = *this*(3) **and** *mid* = *this*(4)  
**show** ?case  
**proof** (*cases* *xs* = LNil)  
**case** *True*  
**then show** ?thesis  
**using** *r\_ys* *mid* **by** *simp* (*rule* *chain\_cons*)  
**next**  
**case** *xs\_nnil*: *False*  
**have** *r\_xs*: *chain* *R* *xs*  
**by** (*metis* *chain.simps* *ltl\_simps*(2) *r\_xxs* *xs\_nnil*)  
**have** *mid'*: *R* (llast *xs*) (lhd *ys*)  
**by** (*metis* *llast\_LCons* *lnull\_def* *mid* *xs\_nnil*)  
**have** *start*: *R* *x* (lhd (*lappend* *xs* *ys*))  
**by** (*metis* (*no\_types*) *chain.simps* *lhd\_LCons* *lhd\_lappend* *chain\_not\_lnull* *ltl\_simps*(2) *r\_xxs* *xs\_nnil*)  
**show** ?thesis  
**unfolding** *lappend\_code*(2) **using** *ih*[OF *r\_xs* *mid'*] *start* **by** (*rule* *chain\_cons*)  
**qed**  
**qed** *simp*  
**qed** (*simp* *add*: *r\_xs* *lappend\_inf*)

**lemma** *chain\_length\_pos*: *chain* *R* *xs* ⇒ *llength* *xs* > 0  
**by** (*cases* *xs*) *simp*+

**lemma** *chain\_ldropn*:

**assumes** *chain* *R* *xs* **and** *enat* *n* < *llength* *xs*  
**shows** *chain* *R* (*ldropn* *n* *xs*)  
**using** *assms*  
**by** (*induct* *n* *arbitrary*: *xs*, *simp*,  
*metis* *chain.cases* *ldrop\_eSuc\_ltl* *ldropn\_LNil* *ldropn\_eq\_LNil* *ltl\_simps*(2) *not\_less*)

**lemma** *chain\_lnth\_rel*:

**assumes**  
*chain*: *chain* *R* *xs* **and**  
*len*: *enat* (*Suc* *j*) < *llength* *xs*  
**shows** *R* (*lnth* *xs* *j*) (*lnth* *xs* (*Suc* *j*))



```

proof –
  define ys where ys = ldropn j xs
  have llength ys > 1
    unfolding ys_def using len
    by (metis One_nat_def funpow_swap1 ldropn_0 ldropn_def ldropn_eq_LNil ldropn_ltl not_less one_enat_def)
  obtain y0 y1 ys' where
    ys: ys = LCons y0 (LCons y1 ys')
    unfolding ys_def by (metis Suc_ile_eq ldropn_Suc_conv_ldropn len less_imp_not_less not_less)
  have chain R ys
    unfolding ys_def using Suc_ile_eq chain chain_ldropn len less_imp_le by blast
  then have R y0 y1
    unfolding ys by (auto elim: chain.cases)
  then show ?thesis
    using ys_def unfolding ys by (metis ldropn_Suc_conv_ldropn ldropn_eq_LConsD llist.inject)
qed

```

```

lemma infinite_chain_lnth_rel:
  assumes  $\neg$  lfinite c and chain r c
  shows r (lnth c i) (lnth c (Suc i))
  using assms chain_lnth_rel lfinite_conv_llength_enat by force

```

```

lemma lnth_rel_chain:
  assumes
     $\neg$  lnull xs and
     $\forall j. \text{enat } (j + 1) < \text{llength } xs \longrightarrow R (\text{lnth } xs \ j) (\text{lnth } xs \ (j + 1))$ 
  shows chain R xs
  using assms
proof (coinduction arbitrary: xs rule: chain.coinduct)
  case chain
  note nnul = this(1) and nth_chain = this(2)

```

```

  show ?case
  proof (cases lnull (ltl xs))
    case True
      have xs = LCons (lhd xs) LNil
      using nnul True by (simp add: llist.expand)
      then show ?thesis
        by blast
    next
      case nnul': False
      moreover have xs = LCons (lhd xs) (ltl xs)
        using nnul by simp
      moreover have
         $\forall j. \text{enat } (j + 1) < \text{llength } (\text{ltl } xs) \longrightarrow R (\text{lnth } (\text{ltl } xs) \ j) (\text{lnth } (\text{ltl } xs) \ (j + 1))$ 
        using nnul nth_chain
        by (metis Suc_eq_plus1 ldrop_eSuc_ltl ldropn_Suc_conv_ldropn ldropn_eq_LConsD lnth_ltl)
      moreover have R (lhd xs) (lhd (ltl xs))
        using nnul' nnul nth_chain[rule_format, of 0, simplified]
        by (metis ldropn_0 ldropn_Suc_conv_ldropn ldropn_eq_LConsD lhd_LCons_ltl lhd_conv_lnth lnth_Suc_LCons ltl_simps(2))
      ultimately show ?thesis
        by blast
  qed
qed

```

```

lemma chain_lmap:
  assumes  $\forall x y. R \ x \ y \longrightarrow R' \ (f \ x) \ (f \ y)$  and chain R xs
  shows chain R' (lmap f xs)
  using assms
proof (coinduction arbitrary: xs)
  case chain
  then have  $(\exists y. xs = LCons \ y \ LNil) \vee (\exists ys \ x. xs = LCons \ x \ ys \wedge \text{chain } R \ ys \wedge R \ x \ (\text{lhd } ys))$ 

```

```

    using chain.simps[of R xs] by auto
  then show ?case
proof
  assume  $\exists ys\ x. xs = LCons\ x\ ys \wedge chain\ R\ ys \wedge R\ x\ (lhd\ ys)$ 
  then have  $\exists ys\ x. lmap\ f\ xs = LCons\ x\ ys \wedge$ 
     $(\exists xs. ys = lmap\ f\ xs \wedge (\forall x\ y. R\ x\ y \longrightarrow R'\ (f\ x)\ (f\ y)) \wedge chain\ R\ xs) \wedge R'\ x\ (lhd\ ys)$ 
  using chain
  by (metis (no_types) lhd_LCons llist.distinct(1) llist.exhaust_sel llist.map_sel(1)
    lmap_eq_LNil chain_not_null ltl_lmap ltl_simps(2))
  then show ?thesis
  by auto
qed auto
qed

lemma chain_mono:
  assumes  $\forall x\ y. R\ x\ y \longrightarrow R'\ x\ y$  and chain R xs
  shows chain R' xs
  using assms by (rule chain_lmap[of _ -  $\lambda x. x$ , unfolded llist.map_ident])

lemma lfinite_chain_imp_rtrancl_lhd_llast:  $lfinite\ xs \implies chain\ R\ xs \implies R^{**}\ (lhd\ xs)\ (llast\ xs)$ 
proof (induct rule: lfinite.induct)
  case (lfinite_LConsI xs x)
  note fin_xs = this(1) and ih = this(2) and r_xs_xs = this(3)
  show ?case
proof (cases xs = LNil)
  case xs_nnil: False
  then have r_xs: chain R xs
    using r_xs_xs by (blast elim: chain.cases)
  then show ?thesis
    using ih[OF r_xs] xs_nnil r_xs_xs
    by (metis chain.cases converse_rtrancl_into_rtrancl lhd_LCons llast_LCons chain_not_null
      ltl_simps(2))
  qed simp
qed simp

lemma trancl_imp_exists_finite_chain_list:
   $R^{++}\ x\ y \implies \exists xs. xs \neq [] \wedge tl\ xs \neq [] \wedge chain\ R\ (llist\_of\ xs) \wedge hd\ xs = x \wedge last\ xs = y$ 
proof (induct rule: trancl.induct)
  case (r_into_trancl x y)
  note r_xy = this

  define xs where
    xs = [x, y]

  have xs  $\neq []$  and tl xs  $\neq []$  and chain R (llist_of xs) and hd xs = x and last xs = y
    unfolding xs_def using r_xy by (auto intro: chain.intros)
  then show ?case
    by blast

next
  case (trancl_into_trancl x y z)
  note rstar_xy = this(1) and ih = this(2) and r_yz = this(3)

  obtain xs where
    xs: xs  $\neq []$  tl xs  $\neq []$  chain R (llist_of xs) hd xs = x last xs = y
    using ih by blast
  define ys where
    ys = xs @ [z]

  have ys  $\neq []$  and tl ys  $\neq []$  and chain R (llist_of ys) and hd ys = x and last ys = z
    unfolding ys_def using xs r_yz
    by (auto simp: lappend_llist_of_llist_of[symmetric] intro: chain_singleton chain_lappend)
  then show ?case
    by blast

```

qed

**inductive-cases** *chain\_consE*: *chain R (LCons x xs)*

**inductive-cases** *chain\_nontrivE*: *chain R (LCons x (LCons y xs))*

**primrec** *prepend* **where**

*prepend [] ys = ys*

| *prepend (x # xs) ys = LCons x (prepend xs ys)*

**lemma** *prepend\_butlast*:

*xs ≠ [] ⟹ ¬ lnull ys ⟹ last xs = lhd ys ⟹ prepend (butlast xs) ys = prepend xs (ltl ys)*

**by** (*induct xs*) *auto*

**lemma** *lnull\_prepend[simp]*: *lnull (prepend xs ys) = (xs = [] ∧ lnull ys)*

**by** (*induct xs*) *auto*

**lemma** *lhd\_prepend[simp]*: *lhd (prepend xs ys) = (if xs ≠ [] then hd xs else lhd ys)*

**by** (*induct xs*) *auto*

**lemma** *prepend\_LNil[simp]*: *prepend xs LNil = llist\_of xs*

**by** (*induct xs*) *auto*

**lemma** *lfinite\_prepend[simp]*: *lfinite (prepend xs ys) ⟷ lfinite ys*

**by** (*induct xs*) *auto*

**lemma** *llength\_prepend[simp]*: *llength (prepend xs ys) = length xs + llength ys*

**by** (*induct xs*) (*auto simp: enat\_0 iadd\_Suc eSuc\_enat[symmetric]*)

**lemma** *llast\_prepend[simp]*: *¬ lnull ys ⟹ llast (prepend xs ys) = llast ys*

**by** (*induct xs*) (*auto simp: llast\_LCons*)

**lemma** *prepend\_prepend*: *prepend xs (prepend ys zs) = prepend (xs @ ys) zs*

**by** (*induct xs*) *auto*

**lemma** *chain\_prepend*:

*chain R (llist\_of zs) ⟹ last zs = lhd xs ⟹ chain R xs ⟹ chain R (prepend zs (ltl xs))*

**by** (*induct zs; cases xs*)

(*auto split: if\_splits simp: lnull\_def[symmetric] intro!: chain\_cons elim!: chain\_consE*)

**lemma** *lmap\_prepend[simp]*: *lmap f (prepend xs ys) = prepend (map f xs) (lmap f ys)*

**by** (*induct xs*) *auto*

**lemma** *lset\_prepend[simp]*: *lset (prepend xs ys) = set xs ∪ lset ys*

**by** (*induct xs*) *auto*

**lemma** *prepend\_LCons*: *prepend xs (LCons y ys) = prepend (xs @ [y]) ys*

**by** (*induct xs*) *auto*

**lemma** *lnth\_prepend*:

*lnth (prepend xs ys) i = (if i < length xs then nth xs i else lnth ys (i - length xs))*

**by** (*induct xs arbitrary: i*) (*auto simp: lnth\_LCons' nth\_Cons'*)

**theorem** *lfinite\_less\_induct[consumes 1, case\_names less]*:

**assumes** *fin: lfinite xs*

**and** *step: ∧xs. lfinite xs ⟹ (∧zs. llength zs < llength xs ⟹ P zs) ⟹ P xs*

**shows** *P xs*

**using** *fin proof* (*induct the\_enat (llength xs) arbitrary: xs rule: less\_induct*)

**case** (*less xs*)

**show** *?case*

**using** *less(2)* **by** (*intro step[OF less(2)] less(1)*)

(*auto dest!: lfinite\_llength\_enat simp: eSuc\_enat elim!: less\_enatE llength\_eq\_enat\_lfiniteD*)

qed

```

theorem lfinite_prepend_induct[consumes 1, case_names LNil prepend]:
  assumes lfinite xs
  and LNil: P LNil
  and prepend:  $\bigwedge xs. lfinite\ xs \implies (\bigwedge zs. (\exists ys. xs = prepend\ ys\ zs \wedge ys \neq [] \implies P\ zs) \implies P\ xs)$ 
  shows P xs
using assms(1) proof (induct xs rule: lfinite_less_induct)
  case (less xs)
  from less(1) show ?case
  by (cases xs)
  (force simp: LNil neq_Nil_conv dest: lfinite_llength_enat intro!: prepend[of LCons _ _] intro: less)+
qed

```

```

coinductive emb :: 'a llist  $\Rightarrow$  'a llist  $\Rightarrow$  bool where
  emb LNil xs
| emb xs ys  $\implies emb (LCons x xs) (prepend zs (LCons x ys))$ 

```

```

inductive prepend_cong1 for X where
  prepend_cong1_base: X xs  $\implies prepend\_cong1\ X\ xs$ 
| prepend_cong1_prepend: prepend\_cong1\ X\ ys  $\implies prepend\_cong1\ X\ (prepend\ xs\ ys)$ 

```

```

lemma emb_prepend_coinduct[rotated, case_names emb]:
  assumes ( $\bigwedge x1\ x2. X\ x1\ x2 \implies$ 
    ( $\exists xs. x1 = LNil \wedge x2 = xs$ )
     $\vee (\exists xs\ ys\ zs. x1 = LCons\ x\ xs \wedge x2 = prepend\ zs\ (LCons\ x\ ys)$ 
       $\wedge (prepend\_cong1\ (X\ xs)\ ys \vee emb\ xs\ ys)))$ ) (is  $\bigwedge x1\ x2. X\ x1\ x2 \implies ?bisim\ x1\ x2$ )
  shows X x1 x2  $\implies emb\ x1\ x2$ 
proof (erule emb.coinduct[OF prepend_cong1_base])
  fix xs zs
  assume prepend_cong1 (X xs) zs
  then show ?bisim xs zs
  by (induct zs rule: prepend_cong1.induct) (erule assms, force simp: prepend_prepend)
qed

```

```

context
begin

```

```

private coinductive chain' for R where
  chain' R (LCons x LNil)
| chain R (llist_of zs  $\implies zs \neq [] \implies tl\ zs \neq [] \implies \neg lnull\ xs \implies last\ zs = lhd\ xs \implies$ 
  ys = ltl xs  $\implies chain'\ R\ xs \implies chain'\ R\ (prepend\ zs\ ys)$ 

```

```

private lemma chain_imp_chain': chain R xs  $\implies chain'\ R\ xs$ 
proof (coinduction arbitrary: xs rule: chain'.coinduct)
  case chain'
  then show ?case
  proof (cases rule: chain.cases)
    case (chain_cons zs z)
    then show ?thesis
    by (intro disjI2 (force intro: chain.intros exI[of _ [z, lhd zs]] exI[of _ zs]
      elim: chain.cases)
    qed simp
qed

```

```

private inductive-cases chain'_LConsE: chain' R (LCons x xs)

```

```

private lemma chain'_stepD1:
  assumes chain' R (LCons x (LCons y xs))
  shows chain' R (LCons y xs)
proof (cases xs)
  case [simp]: (LCons z zs)
  with assms show ?thesis
  proof (cases rule: chain'.cases)
    case (2 as ys xs)

```

```

then show ?thesis
proof (cases tl (tl as))
  case Nil
    with 2 show ?thesis by (auto simp: neq_Nil_conv)
  next
    case (Cons b bs)
    with 2 have chain' R (prepend (y # b # bs) xs)
      by (intro chain'.intros)
      (auto simp: chain_cons not_null_conv neq_Nil_conv elim: chain_nontrivE)
    with 2 Cons show ?thesis
      by (auto simp: neq_Nil_conv)
  qed
qed
qed (simp only: chain'.intros(1))

private lemma chain'_stepD2: chain' R (LCons x (LCons y xs))  $\implies$  R x y
  by (erule chain'.cases) (auto simp: neq_Nil_conv elim!: chain_nontrivE split: if_splits)

private lemma chain'_imp_chain: chain' R xs  $\implies$  chain R xs
proof (coinduction arbitrary: xs rule: chain.coinduct)
  case chain
  then show ?case
  proof (cases rule: chain'.cases)
    case (2 ys zs xs)
    then show ?thesis
    proof (cases ltl zs)
      case LNil
      with chain 2 show ?thesis
        by (auto 0 4 simp: neq_Nil_conv not_null_conv elim: chain'_stepD1 chain'_stepD2)
    next
      case (LCons b bs)
      with chain 2 show ?thesis
        unfolding neq_Nil_conv not_null_conv
        by (elim exE) (auto elim: chain'_stepD1 chain_nontrivE)
    qed
  qed simp
qed

private lemma chain_chain': chain = chain'
  unfolding fun_eq_iff by (metis chain_imp_chain' chain'_imp_chain)

lemma chain_prepend_coinduct[case_names chain]:
   $X\ x \implies (\bigwedge x. X\ x \implies$ 
     $(\exists z. x = LCons\ z\ LNil) \vee$ 
     $(\exists xs\ zs. x = prepend\ zs\ (ltl\ xs) \wedge zs \neq [] \wedge tl\ zs \neq [] \wedge \neg lnull\ xs \wedge last\ zs = lhd\ xs \wedge$ 
     $(X\ xs \vee chain\ R\ xs) \wedge chain\ R\ (llist\_of\ zs))) \implies chain\ R\ x$ 
  by (subst chain_chain', erule chain'.coinduct) (auto simp: chain_chain')

end

context
  fixes R :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool
begin

private definition pick where
  pick x y = (SOME xs. xs  $\neq$  []  $\wedge$  tl xs  $\neq$  []  $\wedge$  chain R (llist_of xs)  $\wedge$  hd xs = x  $\wedge$  last xs = y)

private lemma pick[simp]:
  assumes R++ x y
  shows pick x y  $\neq$  [] tl (pick x y)  $\neq$  [] chain R (llist_of (pick x y))
    hd (pick x y) = x last (pick x y) = y
  unfolding pick_def using tranclp_imp_exists_finite_chain_list[THEN someI_ex, OF assms] by auto

```

**private lemma** *butlast\_pick*[simp]:  $R^{++} \ x \ y \implies \text{butlast} \ (\text{pick} \ x \ y) \neq []$   
**by** (cases *pick*  $x \ y$ ; cases *tl* (*pick*  $x \ y$ )) (auto dest: *pick*(2))

**private friend-of-corec** *prepend* **where**  
*prepend*  $xs \ ys = (\text{case } xs \text{ of } [] \Rightarrow$   
 $(\text{case } ys \text{ of } LNil \Rightarrow LNil \mid LCons \ x \ xs \Rightarrow LCons \ x \ xs) \mid x \ \# \ xs' \Rightarrow LCons \ x \ (\text{prepend } xs' \ ys))$   
**by** (simp split: *list.splits* *llist.splits*) *transfer\_prover*

**private corec** *wit* **where**  
*wit*  $xs = (\text{case } xs \text{ of } LCons \ x \ (LCons \ y \ xs) \Rightarrow$   
 $\text{let } zs = \text{pick } x \ y \text{ in } LCons \ (\text{hd } zs) \ (\text{prepend} \ (\text{butlast} \ (\text{tl } zs)) \ (\text{wit} \ (LCons \ y \ xs))) \mid \_ \Rightarrow xs)$

**private lemma**  
*wit.LNil*[simp]: *wit*  $LNil = LNil$  **and**  
*wit.lsingleton*[simp]: *wit*  $(LCons \ x \ LNil) = LCons \ x \ LNil$  **and**  
*wit.LCons2*: *wit*  $(LCons \ x \ (LCons \ y \ xs)) =$   
 $(\text{let } zs = \text{pick } x \ y \text{ in } LCons \ (\text{hd } zs) \ (\text{prepend} \ (\text{butlast} \ (\text{tl } zs)) \ (\text{wit} \ (LCons \ y \ xs))))$   
**by** (subst *wit.code*; auto)+

**private lemma** *wit.LCons*: *wit*  $(LCons \ x \ xs) = (\text{case } xs \text{ of } LNil \Rightarrow LCons \ x \ LNil \mid LCons \ y \ xs \Rightarrow$   
 $(\text{let } zs = \text{pick } x \ y \text{ in } LCons \ (\text{hd } zs) \ (\text{prepend} \ (\text{butlast} \ (\text{tl } zs)) \ (\text{wit} \ (LCons \ y \ xs))))$   
**by** (subst *wit.code*; auto split: *llist.splits*)+

**private lemma** *null\_wit*[simp]:  $\text{lnull} \ (\text{wit } xs) \longleftrightarrow \text{lnull } xs$   
**by** (subst *wit.code*) (auto split: *llist.splits* simp: *Let\_def*)

**private lemma** *lhd\_wit*[simp]:  $\text{chain } R^{++} \ xs \implies \text{lhd} \ (\text{wit } xs) = \text{lhd } xs$   
**by** (erule *chain.cases*; subst *wit.code*) (auto split: *llist.splits* simp: *Let\_def*)

**private lemma** *butlast\_alt*:  $\text{butlast } xs = (\text{if } \text{tl } xs = [] \text{ then } [] \text{ else } \text{hd } xs \ \# \ \text{butlast} \ (\text{tl } xs))$   
**by** (cases  $xs$ ) auto

**private lemma** *wit\_alt*:  
 $\text{chain } R^{++} \ xs \implies \text{wit } xs = (\text{case } xs \text{ of } LCons \ x \ (LCons \ y \ xs) \Rightarrow$   
 $\text{prepend} \ (\text{pick } x \ y) \ (\text{tl} \ (\text{wit} \ (LCons \ y \ xs))) \mid \_ \Rightarrow xs)$   
**by** (auto split: *llist.splits* simp: *prepend\_butlast*[symmetric] *wit.LCons2* *Let\_def*  
*prepend.simps*(2)[symmetric] *butlast\_alt*[of *pick* - -]  
simp del: *prepend.simps* elim!: *chain\_nontrivE*)

**private lemma** *wit\_alt2*:  
 $\text{chain } R^{++} \ xs \implies \text{wit } xs = (\text{case } xs \text{ of } LCons \ x \ (LCons \ y \ xs) \Rightarrow$   
 $\text{prepend} \ (\text{butlast} \ (\text{pick } x \ y)) \ (\text{wit} \ (LCons \ y \ xs)) \mid \_ \Rightarrow xs)$   
**by** (auto split: *llist.splits* simp: *wit.LCons2* *Let\_def*  
*prepend.simps*(2)[symmetric] *butlast\_alt*[of *pick* - -]  
simp del: *prepend.simps* elim!: *chain\_nontrivE*)

**private lemma** *LNil\_eq\_iff\_lnull*:  $LNil = xs \longleftrightarrow \text{lnull } xs$   
**by** (cases  $xs$ ) auto

**private lemma** *lfinite\_wit*[simp]:  
**assumes**  $\text{chain } R^{++} \ xs$   
**shows**  $\text{lfinite} \ (\text{wit } xs) \longleftrightarrow \text{lfinite } xs$   
**proof**  
**assume** *lfinite* (*wit*  $xs$ )  
**from** *this* **assms** **show** *lfinite*  $xs$   
**proof** (induct *wit*  $xs$  arbitrary:  $xs$  rule: *lfinite\_prepend\_induct*)  
**case** (*prepend*  $zs$ )  
**then show** ?*case*  
**proof** (cases  $zs$ )  
**case** [simp]: ( $LCons \ x \ xs$ )  
**then show** ?*thesis*  
**proof** (cases  $xs$ )  
**case** [simp]:  $LCons$

```

    with prepend show ?thesis
    by (subst (asm) (2) wit_alt2) (force split: llist.splits elim!: chain_nontrivE)+
  qed simp
  qed simp
  qed (simp add: LNil_eq_iff_lnull)
next
  assume lfinite xs
  then show lfinite (wit xs)
  proof (induct xs rule: lfinite.induct)
    case (lfinite_LConsI xs x)
    then show ?case
    by (cases xs) (auto simp: wit_LCons Let_def)
  qed simp
qed

private lemma llast_wit[simp]:
  assumes chain  $R^{++}$  xs
  shows llast (wit xs) = llast xs
proof (cases lfinite xs)
  case True
  from this assms show ?thesis
  proof (induct rule: lfinite.induct)
    case (lfinite_LConsI xs x)
    then show ?case
    by (cases xs) (auto simp: wit_LCons2 llast_LCons elim: chain_nontrivE)
  qed auto
qed (auto simp: llast_linfiniteness assms)

lemma emb_wit[simp]: chain  $R^{++}$  xs  $\implies$  emb xs (wit xs)
proof (coinduction arbitrary: xs rule: emb_prepend_coinduct)
  case (emb xs)
  then show ?case
  proof (cases rule: chain.cases)
    case (chain_cons zs z)
    then show ?thesis
    by (subst (2) wit_code)
      (auto split: llist.splits intro!: exI[of - []] exI[of - :: - llist]
        prepend_cong1_prepend[OF prepend_cong1_base])
  qed (auto intro!: exI[of - LNil] exI[of - []] emb.intros)
qed

lemma chain_transclp_imp_exists_chain:
  chain  $R^{++}$  xs  $\implies$ 
   $\exists$  ys. chain R ys  $\wedge$  emb xs ys  $\wedge$  (lfinite ys  $\longleftrightarrow$  lfinite xs)  $\wedge$  lhd ys = lhd xs
   $\wedge$  llast ys = llast xs
proof (intro exI[of - wit xs] conjI, coinduction arbitrary: xs rule: chain_prepend_coinduct)
  case chain
  then show ?case
  by (subst (1 2) wit_alt; assumption?) (erule chain.cases; force split: llist.splits)
qed auto

inductive-cases emb_LConsE: emb (LCons z zs) ys
inductive-cases emb_LNil2E: emb xs LNil

lemma emb_lset_mono[rotated]:  $x \in \text{lset } xs \implies \text{emb } xs \text{ } ys \implies x \in \text{lset } ys$ 
  by (induct x xs arbitrary: ys rule: llist.set_induct) (auto elim!: emb_LConsE)

lemma emb_Ball_lset_antimono:
  assumes emb Xs Ys
  shows  $\forall Y \in \text{lset } Ys. x \in Y \implies \forall X \in \text{lset } Xs. x \in X$ 
  using emb_lset_mono[OF assms] by blast

lemma emb_lfinite_antimono[rotated]: lfinite ys  $\implies \text{emb } xs \text{ } ys \implies \text{lfinite } xs$ 

```

```

by (induct ys arbitrary: xs rule: lfinite_prepend_induct)
  (force elim!: emb_LNil2E simp: LNil_eq_iff_lnull prepend_LCons elim: emb.cases)+

lemma emb_Liminf_llist_mono_aux:
  assumes emb Xs Ys and  $\neg$  lfinite Xs and  $\neg$  lfinite Ys and  $\forall j \geq i. x \in \text{lnth } Ys\ j$ 
  shows  $\forall j \geq i. x \in \text{lnth } Xs\ j$ 
using assms proof (induct i arbitrary: Xs Ys rule: less_induct)
  case (less i)
  then show ?case
  proof (cases i)
    case 0
    then show ?thesis
      using emb_Ball_lset_antimono[OF less(2), of x] less(5)
      unfolding Ball_def in_lset_conv_lnth simp_thms
      not_lfinite_llength[OF less(3)] not_lfinite_llength[OF less(4)] enat_ord_code subset_eq
      by blast
  next
  case [simp]: (Suc nat)
  from less(2,3) obtain xs as b bs where
    [simp]:  $Xs = LCons\ b\ xs$   $Ys = \text{prepend}\ as\ (LCons\ b\ bs)$  and  $\text{emb}\ xs\ bs$ 
  by (auto elim: emb.cases)
  have IH:  $\forall k \geq j. x \in \text{lnth}\ xs\ k$  if  $\forall k \geq j. x \in \text{lnth}\ bs\ k$   $j < i$  for  $j$ 
  using that less(1)[OF  $\langle \text{emb}\ xs\ bs \rangle$ ] less(3,4) by auto
  from less(5) have  $\forall k \geq i - \text{length}\ as - 1. x \in \text{lnth}\ xs\ k$ 
  by (intro IH allI)
  (drule spec[of  $- + \text{length}\ as + 1$ ], auto simp: lnth_prepend lnth_LCons')
  then show ?thesis
  by (auto simp: lnth_LCons')
qed
qed

lemma emb_Liminf_llist_infinite:
  assumes emb Xs Ys and  $\neg$  lfinite Xs
  shows  $\text{Liminf\_llist } Ys \subseteq \text{Liminf\_llist } Xs$ 
proof -
  from assms have  $\neg$  lfinite Ys
  using emb_lfinite_antimono by blast
  with assms show ?thesis
  unfolding Liminf_llist_def by (auto simp: not_lfinite_llength dest: emb_Liminf_llist_mono_aux)
qed

lemma emb_lmap:  $\text{emb}\ xs\ ys \implies \text{emb}\ (\text{lmap}\ f\ xs)\ (\text{lmap}\ f\ ys)$ 
proof (coinduction arbitrary: xs ys rule: emb.coinduct)
  case emb
  show ?case
  proof (cases xs)
    case xs: (LCons x xs')
    obtain ysa0 and zs0 where
      ys:  $ys = \text{prepend}\ zs0\ (LCons\ x\ ysa0)$  and
      emb':  $\text{emb}\ xs'\ ysa0$ 
    using emb_LConsE[OF emb[unfolded xs]] by metis

    let ?xa = f x
    let ?xsa = lmap f xs'
    let ?zs = map f zs0
    let ?ysa = lmap f ysa0

    have lmap f xs = LCons ?xa ?xsa
      unfolding xs by simp
    moreover have lmap f ys = prepend ?zs (LCons ?xa ?ysa)
      unfolding ys by simp
    moreover have  $\exists xsa\ ysa. ?xsa = \text{lmap}\ f\ xsa \wedge ?ysa = \text{lmap}\ f\ ysa \wedge \text{emb}\ xsa\ ysa$ 

```



```

    using emb' by blast
    ultimately show ?thesis
    by blast
qed simp
qed

end

lemma chain_inf_llist_if_infinite_chain_function:
  assumes  $\forall i. r (f (Suc i)) (f i)$ 
  shows  $\neg lfinite (inf\_llist f) \wedge chain\ r^{-1-1} (inf\_llist f)$ 
  using assms by (simp add: lnth_rel_chain)

lemma infinite_chain_function_iff_infinite_chain_llist:
   $(\exists f. \forall i. r (f (Suc i)) (f i)) \longleftrightarrow (\exists c. \neg lfinite\ c \wedge chain\ r^{-1-1}\ c)$ 
  using chain_inf_llist_if_infinite_chain_function infinite_chain_lnth_rel by blast

lemma wfP_iff_no_infinite_down_chain_llist:  $wfP\ r \longleftrightarrow (\nexists c. \neg lfinite\ c \wedge chain\ r^{-1-1}\ c)$ 
proof -
  have  $wfP\ r \longleftrightarrow wf\ \{(x, y). r\ x\ y\}$ 
  unfolding wfP_def by auto
  also have  $\dots \longleftrightarrow (\nexists f. \forall i. (f (Suc i), f i) \in \{(x, y). r\ x\ y\})$ 
  using wf_iff_no_infinite_down_chain by blast
  also have  $\dots \longleftrightarrow (\nexists f. \forall i. r (f (Suc i)) (f i))$ 
  by auto
  also have  $\dots \longleftrightarrow (\nexists c. \neg lfinite\ c \wedge chain\ r^{-1-1}\ c)$ 
  using infinite_chain_function_iff_infinite_chain_llist by blast
  finally show ?thesis
  by auto
qed

```

## 4.2 Full Chains

**coinductive**  $full\_chain :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$  **for**  $R :: 'a \Rightarrow 'a \Rightarrow bool$  **where**  
 $full\_chain\_singleton: (\forall y. \neg R\ x\ y) \Longrightarrow full\_chain\ R\ (LCons\ x\ LNil)$   
 $| full\_chain\_cons: full\_chain\ R\ xs \Longrightarrow R\ x\ (lhd\ xs) \Longrightarrow full\_chain\ R\ (LCons\ x\ xs)$

**lemma**  
 $full\_chain\_LNil[simp]: \neg full\_chain\ R\ LNil$  **and**  
 $full\_chain\_not\_lnull: full\_chain\ R\ xs \Longrightarrow \neg lnull\ xs$   
**by** (auto elim: full\_chain.cases)

**lemma**  $full\_chain\_ldropn$ :  
**assumes**  $full: full\_chain\ R\ xs$  **and**  $enat\ n < llength\ xs$   
**shows**  $full\_chain\ R\ (ldropn\ n\ xs)$   
**using**  $assms$   
**by** (induct  $n$  arbitrary:  $xs, simp,$   
 $metis\ full\_chain.cases\ ldrop\_eSuc\_ltl\ ldropn\_LNil\ ldropn\_eq\_LNil\ ltl\_simps(2)\ not\_less$ )

**lemma**  $full\_chain\_iff\_chain$ :  
 $full\_chain\ R\ xs \longleftrightarrow chain\ R\ xs \wedge (lfinite\ xs \longrightarrow (\forall y. \neg R\ (llast\ xs)\ y))$   
**proof** (intro  $iffI\ conjI\ impI\ allI; (elim\ conjE)?$ )  
**assume**  $full: full\_chain\ R\ xs$

**show**  $chain: chain\ R\ xs$   
**using**  $full$  **by** (coinduction arbitrary:  $xs$ ) (auto elim: full\_chain.cases)

```

{
  fix y
  assume lfinite xs
  then obtain n where
    suc_n:  $Suc\ n = llength\ xs$ 
  by (metis chain chain_length_pos lessE less_enatE lfinite_conv_llength_enat)
}

```

```

  have full_chain R (ldropn n xs)
  by (rule full_chain_ldropn[OF full]) (use suc_n Suc.ile.eq in force)
  moreover have ldropn n xs = LCons (llast xs) LNil
  using suc_n by (metis enat_le_plus_same(2) enat_ord_simps(2) gen_llength_def
    ldropn_Suc_conv_ldropn ldropn_all lessI llast_ldropn llast_singleton llength_code)
  ultimately show  $\neg R$  (llast xs) y
  by (auto elim: full_chain.cases)
}
next
assume
  chain R xs and
  lfinite xs  $\longrightarrow (\forall y. \neg R$  (llast xs) y)
then show full_chain R xs
  by (coinduction arbitrary: xs) (erule chain.cases, simp, metis lfinite_LConsI llast_LCons)
qed

lemma full_chain_imp_chain: full_chain R xs  $\implies$  chain R xs
  using full_chain_iff_chain by blast

lemma full_chain_length_pos: full_chain R xs  $\implies$  llength xs > 0
  by (fact chain_length_pos[OF full_chain_imp_chain])

lemma full_chain_lnth_rel:
  full_chain R xs  $\implies$  enat (Suc j) < llength xs  $\implies$  R (lnth xs j) (lnth xs (Suc j))
  by (fact chain_lnth_rel[OF full_chain_imp_chain])

inductive-cases full_chain_consE: full_chain R (LCons x xs)
inductive-cases full_chain_nontrivE: full_chain R (LCons x (LCons y xs))

lemma full_chain_tranclp_imp_exists_full_chain:
  assumes full: full_chain  $R^{++}$  xs
  shows  $\exists$  ys. full_chain R ys  $\wedge$  emb xs ys  $\wedge$  lfinite ys = lfinite xs  $\wedge$  lhd ys = lhd xs
     $\wedge$  llast ys = llast xs
proof -
  obtain ys where ys:
    chain R ys emb xs ys lfinite ys = lfinite xs lhd ys = lhd xs llast ys = llast xs
  using full_chain_imp_chain[OF full] chain_tranclp_imp_exists_chain by blast
  have full_chain R ys
  using ys(1,3,5) full unfolding full_chain_iff_chain by auto
  then show ?thesis
  using ys(2-5) by auto
qed

end

```

## 5 Clausal Logic

```

theory Clausal_Logic
  imports Nested_Multisets.Ordinal.Multiset.More
begin

```

Resolution operates on clauses, which are disjunctions of literals. The material formalized here corresponds roughly to Sections 2.1 (“Formulas and Clauses”) of Bachmair and Ganzinger, excluding the formula and term syntax.

### 5.1 Literals

Literals consist of a polarity (positive or negative) and an atom, of type  $'a$ .

```

datatype 'a literal =
  is_pos: Pos (atm_of: 'a)
| Neg (atm_of: 'a)

```

**abbreviation**  $is\_neg :: 'a\ literal \Rightarrow bool$  **where**  
 $is\_neg\ L \equiv \neg\ is\_pos\ L$

**lemma**  $Pos\_atm\_of\_iff[simp]: Pos\ (atm\_of\ L) = L \longleftrightarrow is\_pos\ L$   
**by**  $(cases\ L)\ simp+$

**lemma**  $Neg\_atm\_of\_iff[simp]: Neg\ (atm\_of\ L) = L \longleftrightarrow is\_neg\ L$   
**by**  $(cases\ L)\ simp+$

**lemma**  $set\_literal\_atm\_of: set\_literal\ L = \{atm\_of\ L\}$   
**by**  $(cases\ L)\ simp+$

**lemma**  $ex\_lit\_cases: (\exists\ L.\ P\ L) \longleftrightarrow (\exists\ A.\ P\ (Pos\ A) \vee P\ (Neg\ A))$   
**by**  $(metis\ literal\_exhaust)$

**instantiation**  $literal :: (type)\ uminus$   
**begin**

**definition**  $uminus\_literal :: 'a\ literal \Rightarrow 'a\ literal$  **where**  
 $uminus\ L = (if\ is\_pos\ L\ then\ Neg\ else\ Pos)\ (atm\_of\ L)$

**instance** ..

**end**

**lemma**  
 $uminus\_Pos[simp]: -\ Pos\ A = Neg\ A$  **and**  
 $uminus\_Neg[simp]: -\ Neg\ A = Pos\ A$   
**unfolding**  $uminus\_literal\_def$  **by**  $simp\_all$

**lemma**  $atm\_of\_uminus[simp]: atm\_of\ (-L) = atm\_of\ L$   
**by**  $(case\_tac\ L,\ auto)$

**lemma**  $uminus\_of\_uminus\_id[simp]: -\ (-\ (x :: 'v\ literal)) = x$   
**by**  $(simp\ add: uminus\_literal\_def)$

**lemma**  $uminus\_not\_id[simp]: x \neq -\ (x :: 'v\ literal)$   
**by**  $(case\_tac\ x)\ auto$

**lemma**  $uminus\_not\_id'[simp]: -\ x \neq (x :: 'v\ literal)$   
**by**  $(case\_tac\ x,\ auto)$

**lemma**  $uminus\_eq\_inj[iff]: -\ (a :: 'v\ literal) = -\ b \longleftrightarrow a = b$   
**by**  $(case\_tac\ a;\ case\_tac\ b)\ auto+$

**lemma**  $uminus\_lit\_swap: (a :: 'a\ literal) = -\ b \longleftrightarrow -\ a = b$   
**by**  $auto$

**lemma**  $is\_pos\_neg\_not\_is\_pos: is\_pos\ (-\ L) \longleftrightarrow \neg\ is\_pos\ L$   
**by**  $(cases\ L)\ auto$

**instantiation**  $literal :: (preorder)\ preorder$   
**begin**

**definition**  $less\_literal :: 'a\ literal \Rightarrow 'a\ literal \Rightarrow bool$  **where**  
 $less\_literal\ L\ M \longleftrightarrow atm\_of\ L < atm\_of\ M \vee atm\_of\ L \leq atm\_of\ M \wedge is\_neg\ L < is\_neg\ M$

**definition**  $less\_eq\_literal :: 'a\ literal \Rightarrow 'a\ literal \Rightarrow bool$  **where**  
 $less\_eq\_literal\ L\ M \longleftrightarrow atm\_of\ L < atm\_of\ M \vee atm\_of\ L \leq atm\_of\ M \wedge is\_neg\ L \leq is\_neg\ M$

**instance**

**apply**  $intro\_classes$

**unfolding**  $less\_literal\_def\ less\_eq\_literal\_def$  **by**  $(auto\ intro: order\_trans\ simp: less\_le\_not\_le)$

```

end

instantiation literal :: (order) order
begin

instance
  by intro_classes (auto simp: less_eq_literal_def intro: literal.expand)

end

lemma pos_less_neg[simp]: Pos A < Neg A
  unfolding less_literal_def by simp

lemma pos_less_pos_iff[simp]: Pos A < Pos B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma pos_less_neg_iff[simp]: Pos A < Neg B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_literal_def by (auto simp: less_le_not_le)

lemma neg_less_pos_iff[simp]: Neg A < Pos B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma neg_less_neg_iff[simp]: Neg A < Neg B  $\longleftrightarrow$  A < B
  unfolding less_literal_def by simp

lemma pos_le_neg[simp]: Pos A  $\leq$  Neg A
  unfolding less_eq_literal_def by simp

lemma pos_le_pos_iff[simp]: Pos A  $\leq$  Pos B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_eq_literal_def by (auto simp: less_le_not_le)

lemma pos_le_neg_iff[simp]: Pos A  $\leq$  Neg B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_eq_literal_def by (auto simp: less_imp_le)

lemma neg_le_pos_iff[simp]: Neg A  $\leq$  Pos B  $\longleftrightarrow$  A < B
  unfolding less_eq_literal_def by simp

lemma neg_le_neg_iff[simp]: Neg A  $\leq$  Neg B  $\longleftrightarrow$  A  $\leq$  B
  unfolding less_eq_literal_def by (auto simp: less_imp_le)

lemma leq_imp_less_eq_atm_of: L  $\leq$  M  $\implies$  atm_of L  $\leq$  atm_of M
  unfolding less_eq_literal_def using less_imp_le by blast

instantiation literal :: (linorder) linorder
begin

instance
  apply intro_classes
  unfolding less_eq_literal_def less_literal_def by auto

end

instantiation literal :: (wellorder) wellorder
begin

instance
proof intro_classes
  fix P :: 'a literal  $\Rightarrow$  bool and L :: 'a literal
  assume ih:  $\bigwedge L. (\bigwedge M. M < L \implies P M) \implies P L$ 
  have  $\bigwedge x. (\bigwedge y. y < x \implies P (Pos y) \wedge P (Neg y)) \implies P (Pos x) \wedge P (Neg x)$ 
    by (rule conjI[OF ih ih])
    (auto simp: less_literal_def atm_of_def split: literal.splits intro: ih)

```

**then have**  $\bigwedge A. P (Pos\ A) \wedge P (Neg\ A)$   
**by** (rule less\_induct) blast  
**then show**  $P\ L$   
**by** (cases L) simp+  
**qed**

**end**

## 5.2 Clauses

Clauses are (finite) multisets of literals.

**type-synonym** 'a clause = 'a literal multiset

**abbreviation** map\_clause :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a clause  $\Rightarrow$  'b clause **where**  
 map\_clause f  $\equiv$  image\_mset (map\_literal f)

**abbreviation** rel\_clause :: ('a  $\Rightarrow$  'b  $\Rightarrow$  bool)  $\Rightarrow$  'a clause  $\Rightarrow$  'b clause  $\Rightarrow$  bool **where**  
 rel\_clause R  $\equiv$  rel\_mset (rel\_literal R)

**abbreviation** poss :: 'a multiset  $\Rightarrow$  'a clause **where** poss AA  $\equiv$  {#Pos A. A  $\in$  # AA#}

**abbreviation** negs :: 'a multiset  $\Rightarrow$  'a clause **where** negs AA  $\equiv$  {#Neg A. A  $\in$  # AA#}

**lemma** Max\_in\_lits:  $C \neq \{\#\} \Longrightarrow Max\_mset\ C \in \# C$   
**by** simp

**lemma** Max\_atm\_of\_set\_mset\_commute:  $C \neq \{\#\} \Longrightarrow Max\ (atm\_of\ 'set\_mset\ C) = atm\_of\ (Max\_mset\ C)$   
**by** (rule mono\_Max\_commute[symmetric]) (auto simp: mono\_def less\_eq\_literal\_def)

**lemma** Max\_pos\_neg\_less\_multiset:  
**assumes** max:  $Max\_mset\ C = Pos\ A$  **and** neg:  $Neg\ A \in \# D$   
**shows**  $C < D$

**proof** –

**have**  $Max\_mset\ C < Neg\ A$

**using** max **by** simp

**then show** ?thesis

**using** neg **by** (metis (no\_types) Max\_less\_iff empty\_iff ex\_gt\_imp\_less\_multiset finite\_set\_mset)

**qed**

**lemma** pos\_Max\_imp\_neg\_notin:  $Max\_mset\ C = Pos\ A \Longrightarrow Neg\ A \notin \# C$   
**using** Max\_pos\_neg\_less\_multiset **by** blast

**lemma** less\_eq\_Max\_lit:  $C \neq \{\#\} \Longrightarrow C \leq D \Longrightarrow Max\_mset\ C \leq Max\_mset\ D$

**proof** (unfold less\_eq\_multiset\_HO)

**assume**

$ne: C \neq \{\#\}$  **and**

$ex\_gt: \forall x. count\ D\ x < count\ C\ x \longrightarrow (\exists y > x. count\ C\ y < count\ D\ y)$

**from** ne **have**  $Max\_mset\ C \in \# C$

**by** (fast intro: Max\_in\_lits)

**then have**  $\exists l. l \in \# D \wedge \neg l < Max\_mset\ C$

**using** ex\_gt **by** (metis count\_greater\_zero\_iff count\_inI less\_not\_sym)

**then have**  $\neg Max\_mset\ D < Max\_mset\ C$

**by** (metis Max\_coboundedI[OF finite\_set\_mset] le\_less\_trans)

**then show** ?thesis

**by** simp

**qed**

**definition** atms\_of :: 'a clause  $\Rightarrow$  'a set **where**  
 atms\_of C = atm\_of 'set\_mset C

**lemma** atms\_of\_empty[simp]:  $atms\_of\ \{\#\} = \{\}$   
**unfolding** atms\_of\_def **by** simp

**lemma** atms\_of\_singleton[simp]:  $atms\_of\ \{\#L\# \} = \{atm\_of\ L\}$

**unfolding** *atms\_of\_def* **by** *auto*

**lemma** *atms\_of\_add\_mset[simp]*:  $\text{atms\_of } (\text{add\_mset } a \ A) = \text{insert } (\text{atm\_of } a) (\text{atms\_of } A)$   
**unfolding** *atms\_of\_def* **by** *auto*

**lemma** *atms\_of\_union\_mset[simp]*:  $\text{atms\_of } (A \cup\# B) = \text{atms\_of } A \cup \text{atms\_of } B$   
**unfolding** *atms\_of\_def* **by** *auto*

**lemma** *finite\_atms\_of[iff]*:  $\text{finite } (\text{atms\_of } C)$   
**by** (*simp add: atms\_of\_def*)

**lemma** *atm\_of\_lit\_in\_atms\_of*:  $L \in\# C \implies \text{atm\_of } L \in \text{atms\_of } C$   
**by** (*simp add: atms\_of\_def*)

**lemma** *atms\_of\_plus[simp]*:  $\text{atms\_of } (C + D) = \text{atms\_of } C \cup \text{atms\_of } D$   
**unfolding** *atms\_of\_def* **by** *auto*

**lemma** *in\_atms\_of\_minusD*:  $x \in \text{atms\_of } (A - B) \implies x \in \text{atms\_of } A$   
**by** (*auto simp: atms\_of\_def dest: in\_diffD*)

**lemma** *pos\_lit\_in\_atms\_of*:  $\text{Pos } A \in\# C \implies A \in \text{atms\_of } C$   
**unfolding** *atms\_of\_def* **by** *force*

**lemma** *neg\_lit\_in\_atms\_of*:  $\text{Neg } A \in\# C \implies A \in \text{atms\_of } C$   
**unfolding** *atms\_of\_def* **by** *force*

**lemma** *atm\_imp\_pos\_or\_neg\_lit*:  $A \in \text{atms\_of } C \implies \text{Pos } A \in\# C \vee \text{Neg } A \in\# C$   
**unfolding** *atms\_of\_def image\_def mem\_Collect\_eq* **by** (*metis Neg\_atm\_of\_iff Pos\_atm\_of\_iff*)

**lemma** *atm\_iff\_pos\_or\_neg\_lit*:  $A \in \text{atms\_of } L \longleftrightarrow \text{Pos } A \in\# L \vee \text{Neg } A \in\# L$   
**by** (*auto intro: pos\_lit\_in\_atms\_of neg\_lit\_in\_atms\_of dest: atm\_imp\_pos\_or\_neg\_lit*)

**lemma** *atm\_of\_eq\_atm\_of*:  $\text{atm\_of } L = \text{atm\_of } L' \longleftrightarrow (L = L' \vee L = -L')$   
**by** (*cases L; cases L'*) *auto*

**lemma** *atm\_of\_in\_atm\_of\_set\_iff\_in\_set\_or\_uminus\_in\_set*:  $\text{atm\_of } L \in \text{atm\_of } I \longleftrightarrow (L \in I \vee -L \in I)$   
**by** (*auto intro: rev\_image\_eqI simp: atm\_of\_eq\_atm\_of*)

**lemma** *lits\_subseteq\_imp\_atms\_subseteq*:  $\text{set\_mset } C \subseteq \text{set\_mset } D \implies \text{atms\_of } C \subseteq \text{atms\_of } D$   
**unfolding** *atms\_of\_def* **by** *blast*

**lemma** *atms\_empty\_iff\_empty[iff]*:  $\text{atms\_of } C = \{\} \longleftrightarrow C = \{\#\}$   
**unfolding** *atms\_of\_def image\_def Collect\_empty\_eq* **by** *auto*

**lemma**  
*atms\_of\_poss[simp]*:  $\text{atms\_of } (\text{poss } AA) = \text{set\_mset } AA$  **and**  
*atms\_of\_negs[simp]*:  $\text{atms\_of } (\text{negs } AA) = \text{set\_mset } AA$   
**unfolding** *atms\_of\_def image\_def* **by** *auto*

**lemma** *less\_eq\_Max\_atms\_of*:  $C \neq \{\#\} \implies C \leq D \implies \text{Max } (\text{atms\_of } C) \leq \text{Max } (\text{atms\_of } D)$   
**unfolding** *atms\_of\_def*  
**by** (*metis Max\_atm\_of\_set\_mset\_commute leq\_imp\_less\_eq\_atm\_of less\_eq\_Max\_lit less\_eq\_multiset\_empty\_right*)

**lemma** *le\_multiset\_Max\_in\_imp\_Max*:  
 $\text{Max } (\text{atms\_of } D) = A \implies C \leq D \implies A \in \text{atms\_of } C \implies \text{Max } (\text{atms\_of } C) = A$   
**by** (*metis Max.coboundedI[OF finite\_atms\_of] atms\_of\_def empty\_iff eq\_iff image\_subsetI less\_eq\_Max\_atms\_of set\_mset\_empty subset.Compl\_self\_eq*)

**lemma** *atm\_of\_Max\_lit[simp]*:  $C \neq \{\#\} \implies \text{atm\_of } (\text{Max\_mset } C) = \text{Max } (\text{atms\_of } C)$   
**unfolding** *atms\_of\_def Max\_atm\_of\_set\_mset\_commute* **..**

**lemma** *Max\_lit\_eq\_pos\_or\_neg\_Max\_atm*:

$C \neq \{\#\} \implies \text{Max\_mset } C = \text{Pos } (\text{Max } (\text{atms\_of } C)) \vee \text{Max\_mset } C = \text{Neg } (\text{Max } (\text{atms\_of } C))$   
**by** (metis Neg\_atm\_of\_iff Pos\_atm\_of\_iff atm\_of\_Max\_lit)

**lemma** *atms\_less\_imp\_lit\_less\_pos*:  $(\bigwedge B. B \in \text{atms\_of } C \implies B < A) \implies L \in \# C \implies L < \text{Pos } A$   
**unfolding** *atms\_of\_def less\_literal\_def* **by** force

**lemma** *atms\_less\_eq\_imp\_lit\_less\_eq\_neg*:  $(\bigwedge B. B \in \text{atms\_of } C \implies B \leq A) \implies L \in \# C \implies L \leq \text{Neg } A$   
**unfolding** *less\_eq\_literal\_def* **by** (simp add: atm\_of\_lit\_in\_atms\_of)

**end**

## 6 Herbrand Interpretation

**theory** *Herbrand Interpretation*  
**imports** *Clausal Logic*  
**begin**

The material formalized here corresponds roughly to Sections 2.2 (“Herbrand Interpretations”) of Bachmair and Ganzinger, excluding the formula and term syntax.

A Herbrand interpretation is a set of ground atoms that are to be considered true.

**type-synonym** *'a interp* = *'a set*

**definition** *true\_lit* :: *'a interp*  $\Rightarrow$  *'a literal*  $\Rightarrow$  *bool* (**infix**  $\models_l$  50) **where**  
 $I \models_l L \longleftrightarrow (\text{if is\_pos } L \text{ then } (\lambda P. P) \text{ else Not } (\text{atm\_of } L \in I))$

**lemma** *true\_lit\_simps*[*simp*]:  
 $I \models_l \text{Pos } A \longleftrightarrow A \in I$   
 $I \models_l \text{Neg } A \longleftrightarrow A \notin I$   
**unfolding** *true\_lit\_def* **by** auto

**lemma** *true\_lit\_iff*[*iff*]:  $I \models_l L \longleftrightarrow (\exists A. L = \text{Pos } A \wedge A \in I \vee L = \text{Neg } A \wedge A \notin I)$   
**by** (cases *L*) *simp*+

**definition** *true\_cls* :: *'a interp*  $\Rightarrow$  *'a clause*  $\Rightarrow$  *bool* (**infix**  $\models$  50) **where**  
 $I \models C \longleftrightarrow (\exists L \in \# C. I \models_l L)$

**lemma** *true\_cls\_empty*[*iff*]:  $\neg I \models \{\#\}$   
**unfolding** *true\_cls\_def* **by** *simp*

**lemma** *true\_cls\_singleton*[*iff*]:  $I \models \{\#L\} \longleftrightarrow I \models_l L$   
**unfolding** *true\_cls\_def* **by** *simp*

**lemma** *true\_cls\_add\_mset*[*iff*]:  $I \models \text{add\_mset } C D \longleftrightarrow I \models_l C \vee I \models D$   
**unfolding** *true\_cls\_def* **by** auto

**lemma** *true\_cls\_union*[*iff*]:  $I \models C + D \longleftrightarrow I \models C \vee I \models D$   
**unfolding** *true\_cls\_def* **by** auto

**lemma** *true\_cls\_mono*:  $\text{set\_mset } C \subseteq \text{set\_mset } D \implies I \models C \implies I \models D$   
**unfolding** *true\_cls\_def subset\_eq* **by** metis

**lemma**  
**assumes**  $I \subseteq J$   
**shows**  
*false\_to\_true\_imp\_ex\_pos*:  $\neg I \models C \implies J \models C \implies \exists A \in J. \text{Pos } A \in \# C$  **and**  
*true\_to\_false\_imp\_ex\_neg*:  $I \models C \implies \neg J \models C \implies \exists A \in J. \text{Neg } A \in \# C$   
**using** *assms* **unfolding** *subset\_iff true\_cls\_def* **by** (metis *literal.collapse true\_lit\_simps*)+

**lemma** *true\_cls\_replicate\_mset*[*iff*]:  $I \models \text{replicate\_mset } n L \longleftrightarrow n \neq 0 \wedge I \models_l L$   
**by** (simp add: *true\_cls\_def*)

**lemma** *pos\_literal\_in\_imp\_true\_cls*[*intro*]:  $\text{Pos } A \in \# C \implies A \in I \implies I \models C$

```

using true_cls_def by blast

lemma neg_literal_notin_imp_true_cls[intro]: Neg A ∈# C ⇒ A ∉ I ⇒ I ⊨ C
  using true_cls_def by blast

lemma pos_neg_in_imp_true: Pos A ∈# C ⇒ Neg A ∈# C ⇒ I ⊨ C
  using true_cls_def by blast

definition true_clss :: 'a interp ⇒ 'a clause set ⇒ bool (infix ⊨s 50) where
  I ⊨s CC ↔ (∀ C ∈ CC. I ⊨ C)

lemma true_clss_empty[iff]: I ⊨s {}
  by (simp add: true_clss_def)

lemma true_clss_singleton[iff]: I ⊨s {C} ↔ I ⊨ C
  unfolding true_clss_def by blast

lemma true_clss_insert[iff]: I ⊨s insert C DD ↔ I ⊨ C ∧ I ⊨s DD
  unfolding true_clss_def by blast

lemma true_clss_union[iff]: I ⊨s CC ∪ DD ↔ I ⊨s CC ∧ I ⊨s DD
  unfolding true_clss_def by blast

lemma true_clss_mono: DD ⊆ CC ⇒ I ⊨s CC ⇒ I ⊨s DD
  by (simp add: set_mp true_clss_def)

abbreviation satisfiable :: 'a clause set ⇒ bool where
  satisfiable CC ≡ ∃ I. I ⊨s CC

definition true_cls_mset :: 'a interp ⇒ 'a clause multiset ⇒ bool (infix ⊨m 50) where
  I ⊨m CC ↔ (∀ C ∈# CC. I ⊨ C)

lemma true_cls_mset_empty[iff]: I ⊨m {}
  unfolding true_cls_mset_def by auto

lemma true_cls_mset_singleton[iff]: I ⊨m {#C#} ↔ I ⊨ C
  by (simp add: true_cls_mset_def)

lemma true_cls_mset_union[iff]: I ⊨m CC + DD ↔ I ⊨m CC ∧ I ⊨m DD
  unfolding true_cls_mset_def by auto

lemma true_cls_mset_add_mset[iff]: I ⊨m add_mset C CC ↔ I ⊨ C ∧ I ⊨m CC
  unfolding true_cls_mset_def by auto

lemma true_cls_mset_image_mset[iff]: I ⊨m image_mset f A ↔ (∀ x ∈# A. I ⊨ f x)
  unfolding true_cls_mset_def by auto

lemma true_cls_mset_mono: set_mset DD ⊆ set_mset CC ⇒ I ⊨m CC ⇒ I ⊨m DD
  unfolding true_cls_mset_def subset_iff by auto

lemma true_clss_set_mset[iff]: I ⊨s set_mset CC ↔ I ⊨m CC
  unfolding true_clss_def true_cls_mset_def by auto

lemma true_cls_mset_true_cls: I ⊨m CC ⇒ C ∈# CC ⇒ I ⊨ C
  using true_cls_mset_def by auto

end

```

## 7 Abstract Substitutions

```

theory Abstract_Substitution
  imports Clausal_Logic Map2
begin

```



Atoms and substitutions are abstracted away behind some locales, to avoid having a direct dependency on the IsaFoR library.

Conventions:  $'s$  substitutions,  $'a$  atoms.

## 7.1 Library

```

lemma f_Suc_decr_eventually_const:
  fixes  $f :: nat \Rightarrow nat$ 
  assumes  $leq: \forall i. f (Suc\ i) \leq f\ i$ 
  shows  $\exists l. \forall l' \geq l. f\ l' = f (Suc\ l')$ 
proof (rule ccontr)
  assume  $a: \nexists l. \forall l' \geq l. f\ l' = f (Suc\ l')$ 
  have  $\forall i. \exists i'. i' > i \wedge f\ i' < f\ i$ 
  proof
    fix  $i$ 
    from  $a$  have  $\exists l' \geq i. f\ l' \neq f (Suc\ l')$ 
    by auto
    then obtain  $l'$  where
       $l'_p: l' \geq i \wedge f\ l' \neq f (Suc\ l')$ 
    by metis
    then have  $f\ l' > f (Suc\ l')$ 
    using  $leq\ le\_eq\_less\_or\_eq$  by auto
    moreover have  $f\ i \geq f\ l'$ 
    using  $leq\ l'_p$  by (induction  $l'$  arbitrary: i) (blast intro: lift_Suc_antimono_le)+
    ultimately show  $\exists i' > i. f\ i' < f\ i$ 
    using  $l'_p\ less\_le\_trans$  by blast
  qed
then obtain  $g\_sm :: nat \Rightarrow nat$  where
   $g\_sm\_p: \forall i. g\_sm\ i > i \wedge f (g\_sm\ i) < f\ i$ 
  by metis

define  $c :: nat \Rightarrow nat$  where
   $\bigwedge n. c\ n = (g\_sm\ ^\wedge n)\ 0$ 

have  $f (c\ i) > f (c (Suc\ i))$  for  $i$ 
  by (induction i) (auto simp: c_def g_sm_p)
then have  $\forall i. (f \circ c)\ i > (f \circ c) (Suc\ i)$ 
  by auto
then have  $\exists fc :: nat \Rightarrow nat. \forall i. fc\ i > fc (Suc\ i)$ 
  by metis
then show False
  using wf_less_than by (simp add: wf_iff_no_infinite_down_chain)
qed

```

## 7.2 Substitution Operators

```

locale substitution_ops =
  fixes
     $subst\_atm :: 'a \Rightarrow 's \Rightarrow 'a$  and
     $id\_subst :: 's$  and
     $comp\_subst :: 's \Rightarrow 's \Rightarrow 's$ 
begin

abbreviation  $subst\_atm\_abbrev :: 'a \Rightarrow 's \Rightarrow 'a$  (infixl  $\cdot_a$  67) where
   $subst\_atm\_abbrev \equiv subst\_atm$ 

abbreviation  $comp\_subst\_abbrev :: 's \Rightarrow 's \Rightarrow 's$  (infixl  $\odot$  67) where
   $comp\_subst\_abbrev \equiv comp\_subst$ 

definition  $comp\_substs :: 's\ list \Rightarrow 's\ list \Rightarrow 's\ list$  (infixl  $\odot_s$  67) where
   $\sigma s \odot_s \tau s = map2\ comp\_subst\ \sigma s\ \tau s$ 

definition  $subst\_atms :: 'a\ set \Rightarrow 's \Rightarrow 'a\ set$  (infixl  $\cdot_{as}$  67) where

```

$$AA \cdot as \ \sigma = (\lambda A. A \cdot a \ \sigma) \ 'AA$$

**definition** *subst\_atmss* :: 'a set set  $\Rightarrow$  's  $\Rightarrow$  'a set set (**infixl** ·ass 67) **where**  
 $AAA \cdot ass \ \sigma = (\lambda AAA. AA \cdot as \ \sigma) \ 'AAA$

**definition** *subst\_atm\_list* :: 'a list  $\Rightarrow$  's  $\Rightarrow$  'a list (**infixl** ·al 67) **where**  
 $As \cdot al \ \sigma = map \ (\lambda A. A \cdot a \ \sigma) \ As$

**definition** *subst\_atm\_mset* :: 'a multiset  $\Rightarrow$  's  $\Rightarrow$  'a multiset (**infixl** ·am 67) **where**  
 $AA \cdot am \ \sigma = image\_mset \ (\lambda A. A \cdot a \ \sigma) \ AA$

**definition**

$$subst\_atm\_mset\_list :: 'a multiset list \Rightarrow 's \Rightarrow 'a multiset list \ (\mathbf{infixl} \cdot aml \ 67)$$

**where**

$$AAA \cdot aml \ \sigma = map \ (\lambda AAA. AA \cdot am \ \sigma) \ AAA$$

**definition**

$$subst\_atm\_mset\_lists :: 'a multiset list \Rightarrow 's list \Rightarrow 'a multiset list \ (\mathbf{infixl} \cdot aml \ 67)$$

**where**

$$AAs \cdot aml \ \sigma s = map2 \ (op \cdot am) \ AAs \ \sigma s$$

**definition** *subst\_lit* :: 'a literal  $\Rightarrow$  's  $\Rightarrow$  'a literal (**infixl** ·l 67) **where**  
 $L \cdot l \ \sigma = map\_literal \ (\lambda A. A \cdot a \ \sigma) \ L$

**lemma** *atm\_of\_subst\_lit[simp]*:  $atm\_of \ (L \cdot l \ \sigma) = atm\_of \ L \cdot a \ \sigma$   
**unfolding** *subst\_lit\_def* **by** (cases L) *simp*+

**definition** *subst\_cls* :: 'a clause  $\Rightarrow$  's  $\Rightarrow$  'a clause (**infixl** · 67) **where**  
 $AA \cdot \sigma = image\_mset \ (\lambda A. A \cdot l \ \sigma) \ AA$

**definition** *subst\_cls* :: 'a clause set  $\Rightarrow$  's  $\Rightarrow$  'a clause set (**infixl** ·cs 67) **where**  
 $AA \cdot cs \ \sigma = (\lambda A. A \cdot \sigma) \ 'AA$

**definition** *subst\_cls\_list* :: 'a clause list  $\Rightarrow$  's  $\Rightarrow$  'a clause list (**infixl** ·cl 67) **where**  
 $Cs \cdot cl \ \sigma = map \ (\lambda A. A \cdot \sigma) \ Cs$

**definition** *subst\_cls\_lists* :: 'a clause list  $\Rightarrow$  's list  $\Rightarrow$  'a clause list (**infixl** ·cl 67) **where**  
 $Cs \cdot cl \ \sigma s = map2 \ (op \cdot) \ Cs \ \sigma s$

**definition** *subst\_cls\_mset* :: 'a clause multiset  $\Rightarrow$  's  $\Rightarrow$  'a clause multiset (**infixl** ·cm 67) **where**  
 $CC \cdot cm \ \sigma = image\_mset \ (\lambda A. A \cdot \sigma) \ CC$

**lemma** *subst\_cls\_add\_mset[simp]*:  $add\_mset \ L \ C \cdot \sigma = add\_mset \ (L \cdot l \ \sigma) \ (C \cdot \sigma)$   
**unfolding** *subst\_cls\_def* **by** *simp*

**lemma** *subst\_cls\_mset\_add\_mset[simp]*:  $add\_mset \ C \ CC \cdot cm \ \sigma = add\_mset \ (C \cdot \sigma) \ (CC \cdot cm \ \sigma)$   
**unfolding** *subst\_cls\_mset\_def* **by** *simp*

**definition** *generalizes\_atm* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool **where**  
 $generalizes\_atm \ A \ B \longleftrightarrow (\exists \sigma. A \cdot a \ \sigma = B)$

**definition** *strictly\_generalizes\_atm* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool **where**  
 $strictly\_generalizes\_atm \ A \ B \longleftrightarrow generalizes\_atm \ A \ B \wedge \neg generalizes\_atm \ B \ A$

**definition** *generalizes\_lit* :: 'a literal  $\Rightarrow$  'a literal  $\Rightarrow$  bool **where**  
 $generalizes\_lit \ L \ M \longleftrightarrow (\exists \sigma. L \cdot l \ \sigma = M)$

**definition** *strictly\_generalizes\_lit* :: 'a literal  $\Rightarrow$  'a literal  $\Rightarrow$  bool **where**  
 $strictly\_generalizes\_lit \ L \ M \longleftrightarrow generalizes\_lit \ L \ M \wedge \neg generalizes\_lit \ M \ L$

**definition** *generalizes\_cls* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
 $generalizes\_cls \ C \ D \longleftrightarrow (\exists \sigma. C \cdot \sigma = D)$

**definition** *strictly\_generalizes\_cls* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*strictly\_generalizes\_cls* C D  $\longleftrightarrow$  generalizes\_cls C D  $\wedge$   $\neg$  generalizes\_cls D C

**definition** *subsumes* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*subsumes* C D  $\longleftrightarrow$  ( $\exists \sigma. C \cdot \sigma \subseteq_{\#} D$ )

**definition** *strictly\_subsumes* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*strictly\_subsumes* C D  $\longleftrightarrow$  subsumes C D  $\wedge$   $\neg$  subsumes D C

**definition** *variants* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*variants* C D  $\longleftrightarrow$  generalizes\_cls C D  $\wedge$  generalizes\_cls D C

**definition** *is\_renaming* :: 's  $\Rightarrow$  bool **where**  
*is\_renaming*  $\sigma \longleftrightarrow$  ( $\exists \tau. \sigma \odot \tau = id\_subst$ )

**definition** *is\_renaming\_list* :: 's list  $\Rightarrow$  bool **where**  
*is\_renaming\_list*  $\sigma s \longleftrightarrow$  ( $\forall \sigma \in set \sigma s. is\_renaming \sigma$ )

**definition** *inv\_renaming* :: 's  $\Rightarrow$  's **where**  
*inv\_renaming*  $\sigma = (SOME \tau. \sigma \odot \tau = id\_subst)$

**definition** *is\_ground\_atm* :: 'a  $\Rightarrow$  bool **where**  
*is\_ground\_atm* A  $\longleftrightarrow$  ( $\forall \sigma. A = A \cdot a \sigma$ )

**definition** *is\_ground\_atms* :: 'a set  $\Rightarrow$  bool **where**  
*is\_ground\_atms* AA = ( $\forall A \in AA. is\_ground\_atm A$ )

**definition** *is\_ground\_atm\_list* :: 'a list  $\Rightarrow$  bool **where**  
*is\_ground\_atm\_list* As  $\longleftrightarrow$  ( $\forall A \in set As. is\_ground\_atm A$ )

**definition** *is\_ground\_atm\_mset* :: 'a multiset  $\Rightarrow$  bool **where**  
*is\_ground\_atm\_mset* AA  $\longleftrightarrow$  ( $\forall A. A \in_{\#} AA \longrightarrow is\_ground\_atm A$ )

**definition** *is\_ground\_lit* :: 'a literal  $\Rightarrow$  bool **where**  
*is\_ground\_lit* L  $\longleftrightarrow is\_ground\_atm (atm\_of L)$

**definition** *is\_ground\_cls* :: 'a clause  $\Rightarrow$  bool **where**  
*is\_ground\_cls* C  $\longleftrightarrow$  ( $\forall L. L \in_{\#} C \longrightarrow is\_ground\_lit L$ )

**definition** *is\_ground\_cls* :: 'a clause set  $\Rightarrow$  bool **where**  
*is\_ground\_cls* CC  $\longleftrightarrow$  ( $\forall C \in CC. is\_ground\_cls C$ )

**definition** *is\_ground\_cls\_list* :: 'a clause list  $\Rightarrow$  bool **where**  
*is\_ground\_cls\_list* CC  $\longleftrightarrow$  ( $\forall C \in set CC. is\_ground\_cls C$ )

**definition** *is\_ground\_subst* :: 's  $\Rightarrow$  bool **where**  
*is\_ground\_subst*  $\sigma \longleftrightarrow$  ( $\forall A. is\_ground\_atm (A \cdot a \sigma)$ )

**definition** *is\_ground\_subst\_list* :: 's list  $\Rightarrow$  bool **where**  
*is\_ground\_subst\_list*  $\sigma s \longleftrightarrow$  ( $\forall \sigma \in set \sigma s. is\_ground\_subst \sigma$ )

**definition** *grounding\_of\_cls* :: 'a clause  $\Rightarrow$  'a clause set **where**  
*grounding\_of\_cls* C =  $\{C \cdot \sigma \mid \sigma. is\_ground\_subst \sigma\}$

**definition** *grounding\_of\_cls* :: 'a clause set  $\Rightarrow$  'a clause set **where**  
*grounding\_of\_cls* CC = ( $\bigcup C \in CC. grounding\_of\_cls C$ )

**definition** *is\_unifier* :: 's  $\Rightarrow$  'a set  $\Rightarrow$  bool **where**  
*is\_unifier*  $\sigma AA \longleftrightarrow card (AA \cdot as \sigma) \leq 1$

**definition** *is\_unifiers* :: 's  $\Rightarrow$  'a set set  $\Rightarrow$  bool **where**  
*is\_unifiers*  $\sigma AAA \longleftrightarrow$  ( $\forall AA \in AAA. is\_unifier \sigma AA$ )

**definition** *is\_mgu* :: 's  $\Rightarrow$  'a set set  $\Rightarrow$  bool **where**  
*is\_mgu*  $\sigma$  AAA  $\longleftrightarrow$  *is\_unifiers*  $\sigma$  AAA  $\wedge$  ( $\forall \tau. \text{is\_unifiers } \tau \text{ AAA} \longrightarrow (\exists \gamma. \tau = \sigma \odot \gamma)$ )

**definition** *var\_disjoint* :: 'a clause list  $\Rightarrow$  bool **where**  
*var\_disjoint* Cs  $\longleftrightarrow$   
 $(\forall \sigma s. \text{length } \sigma s = \text{length } Cs \longrightarrow (\exists \tau. \forall i < \text{length } Cs. \forall S. S \subseteq_{\#} Cs ! i \longrightarrow S \cdot \sigma s ! i = S \cdot \tau))$

**end**

### 7.3 Substitution Lemmas

**locale** *substitution* = *substitution\_ops subst\_atm id\_subst comp\_subst*  
**for**  
*subst\_atm* :: 'a  $\Rightarrow$  's  $\Rightarrow$  'a **and**  
*id\_subst* :: 's **and**  
*comp\_subst* :: 's  $\Rightarrow$  's  $\Rightarrow$  's +  
**fixes**  
*atm\_of\_atms* :: 'a list  $\Rightarrow$  'a **and**  
*renamings\_apart* :: 'a clause list  $\Rightarrow$  's list  
**assumes**  
*subst\_atm\_id\_subst[simp]*:  $A \cdot a \text{ id\_subst} = A$  **and**  
*subst\_atm\_comp\_subst[simp]*:  $A \cdot a (\tau \odot \sigma) = (A \cdot a \tau) \cdot a \sigma$  **and**  
*subst\_ext*:  $(\bigwedge A. A \cdot a \sigma = A \cdot a \tau) \implies \sigma = \tau$  **and**  
*make\_ground\_subst*:  $\text{is\_ground\_cls } (C \cdot \sigma) \implies \exists \tau. \text{is\_ground\_subst } \tau \wedge C \cdot \tau = C \cdot \sigma$  **and**  
*renames\_apart*:  
 $\bigwedge Cs. \text{length } (\text{renamings\_apart } Cs) = \text{length } Cs \wedge$   
 $(\forall \varrho \in \text{set } (\text{renamings\_apart } Cs). \text{is\_renaming } \varrho) \wedge$   
 $\text{var\_disjoint } (Cs \cdot \text{cl } (\text{renamings\_apart } Cs))$  **and**  
*atm\_of\_atms\_subst*:  
 $\bigwedge As Bs. \text{atm\_of\_atms } As \cdot a \sigma = \text{atm\_of\_atms } Bs \longleftrightarrow \text{map } (\lambda A. A \cdot a \sigma) As = Bs$  **and**  
*wf\_strictly\_generalizes\_atm*: *wfP strictly\_generalizes\_atm*  
**begin**

**lemma** *subst\_ext\_iff*:  $\sigma = \tau \longleftrightarrow (\forall A. A \cdot a \sigma = A \cdot a \tau)$   
**by** (*blast intro: subst\_ext*)

#### 7.3.1 Identity Substitution

**lemma** *id\_subst\_comp\_subst[simp]*:  $\text{id\_subst} \odot \sigma = \sigma$   
**by** (*rule subst\_ext*) *simp*

**lemma** *comp\_subst\_id\_subst[simp]*:  $\sigma \odot \text{id\_subst} = \sigma$   
**by** (*rule subst\_ext*) *simp*

**lemma** *id\_subst\_comp\_substs[simp]*:  $\text{replicate } (\text{length } \sigma s) \text{ id\_subst} \odot s \sigma s = \sigma s$   
**using** *comp\_substs\_def* **by** (*induction*  $\sigma s$ ) *auto*

**lemma** *comp\_substs\_id\_subst[simp]*:  $\sigma s \odot s \text{ replicate } (\text{length } \sigma s) \text{ id\_subst} = \sigma s$   
**using** *comp\_substs\_def* **by** (*induction*  $\sigma s$ ) *auto*

**lemma** *subst\_atms\_id\_subst[simp]*:  $AA \cdot as \text{ id\_subst} = AA$   
**unfolding** *subst\_atms\_def* **by** *simp*

**lemma** *subst\_atmss\_id\_subst[simp]*:  $AAA \cdot ass \text{ id\_subst} = AAA$   
**unfolding** *subst\_atmss\_def* **by** *simp*

**lemma** *subst\_atm\_list\_id\_subst[simp]*:  $As \cdot al \text{ id\_subst} = As$   
**unfolding** *subst\_atm\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_id\_subst[simp]*:  $AA \cdot am \text{ id\_subst} = AA$   
**unfolding** *subst\_atm\_mset\_def* **by** *simp*

**lemma** *subst\_atm\_mset\_list\_id\_subst[simp]*:  $AAs \cdot aml \text{ id\_subst} = AAs$

**unfolding** *subst\_atm\_mset\_list\_def* **by** *simp*

**lemma** *subst\_atm\_mset\_lists\_id\_subst[simp]*:  $AAs \cdot aml \text{ replicate } (\text{length } AAs) \text{ id\_subst} = AAs$   
**unfolding** *subst\_atm\_mset\_lists\_def* **by** (*induct AAs*) *auto*

**lemma** *subst\_lit\_id\_subst[simp]*:  $L \cdot l \text{ id\_subst} = L$   
**unfolding** *subst\_lit\_def* **by** (*simp add: literal.map\_ident*)

**lemma** *subst\_cls\_id\_subst[simp]*:  $C \cdot \text{id\_subst} = C$   
**unfolding** *subst\_cls\_def* **by** *simp*

**lemma** *subst\_cls\_id\_subst[simp]*:  $CC \cdot cs \text{ id\_subst} = CC$   
**unfolding** *subst\_cls\_def* **by** *simp*

**lemma** *subst\_cls\_list\_id\_subst[simp]*:  $Cs \cdot cl \text{ id\_subst} = Cs$   
**unfolding** *subst\_cls\_list\_def* **by** *simp*

**lemma** *subst\_cls\_lists\_id\_subst[simp]*:  $Cs \cdot cl \text{ replicate } (\text{length } Cs) \text{ id\_subst} = Cs$   
**unfolding** *subst\_cls\_lists\_def* **by** (*induct Cs*) *auto*

**lemma** *subst\_cls\_mset\_id\_subst[simp]*:  $CC \cdot cm \text{ id\_subst} = CC$   
**unfolding** *subst\_cls\_mset\_def* **by** *simp*

### 7.3.2 Associativity of Composition

**lemma** *comp\_subst\_assoc[simp]*:  $\sigma \odot (\tau \odot \gamma) = \sigma \odot \tau \odot \gamma$   
**by** (*rule subst\_ext*) *simp*

### 7.3.3 Compatibility of Substitution and Composition

**lemma** *subst\_atms\_comp\_subst[simp]*:  $AA \cdot as (\tau \odot \sigma) = AA \cdot as \tau \cdot as \sigma$   
**unfolding** *subst\_atms\_def* **by** *auto*

**lemma** *subst\_atmss\_comp\_subst[simp]*:  $AAA \cdot ass (\tau \odot \sigma) = AAA \cdot ass \tau \cdot ass \sigma$   
**unfolding** *subst\_atmss\_def* **by** *auto*

**lemma** *subst\_atm\_list\_comp\_subst[simp]*:  $As \cdot al (\tau \odot \sigma) = As \cdot al \tau \cdot al \sigma$   
**unfolding** *subst\_atm\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_comp\_subst[simp]*:  $AA \cdot am (\tau \odot \sigma) = AA \cdot am \tau \cdot am \sigma$   
**unfolding** *subst\_atm\_mset\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_list\_comp\_subst[simp]*:  $AAs \cdot aml (\tau \odot \sigma) = (AAs \cdot aml \tau) \cdot aml \sigma$   
**unfolding** *subst\_atm\_mset\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_lists\_comp\_substs[simp]*:  $AAs \cdot aml (\tau s \odot s \sigma s) = AAs \cdot aml \tau s \cdot aml \sigma s$   
**unfolding** *subst\_atm\_mset\_lists\_def comp\_substs\_def map\_zip\_map map\_zip\_map2 map\_zip\_assoc*  
**by** (*simp add: split\_def*)

**lemma** *subst\_lit\_comp\_subst[simp]*:  $L \cdot l (\tau \odot \sigma) = L \cdot l \tau \cdot l \sigma$   
**unfolding** *subst\_lit\_def* **by** (*auto simp: literal.map\_comp o\_def*)

**lemma** *subst\_cls\_comp\_subst[simp]*:  $C \cdot (\tau \odot \sigma) = C \cdot \tau \cdot \sigma$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_clscomp\_subst[simp]*:  $CC \cdot cs (\tau \odot \sigma) = CC \cdot cs \tau \cdot cs \sigma$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_list\_comp\_subst[simp]*:  $Cs \cdot cl (\tau \odot \sigma) = Cs \cdot cl \tau \cdot cl \sigma$   
**unfolding** *subst\_cls\_list\_def* **by** *auto*

**lemma** *subst\_cls\_lists\_comp\_substs[simp]*:  $Cs \cdot cl (\tau s \odot s \sigma s) = Cs \cdot cl \tau s \cdot cl \sigma s$   
**unfolding** *subst\_cls\_lists\_def comp\_substs\_def map\_zip\_map map\_zip\_map2 map\_zip\_assoc*  
**by** (*simp add: split\_def*)

**lemma** *subst\_cls\_mset\_comp\_subst*[simp]:  $CC \cdot cm (\tau \odot \sigma) = CC \cdot cm \tau \cdot cm \sigma$   
**unfolding** *subst\_cls\_mset\_def* **by** *auto*

### 7.3.4 “Commutativity” of Membership and Substitution

**lemma** *Melem\_subst\_atm\_mset*[simp]:  $A \in\# AA \cdot am \sigma \longleftrightarrow (\exists B. B \in\# AA \wedge A = B \cdot a \sigma)$   
**unfolding** *subst\_atm\_mset\_def* **by** *auto*

**lemma** *Melem\_subst\_cls*[simp]:  $L \in\# C \cdot \sigma \longleftrightarrow (\exists M. M \in\# C \wedge L = M \cdot l \sigma)$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *Melem\_subst\_cls\_mset*[simp]:  $AA \in\# CC \cdot cm \sigma \longleftrightarrow (\exists BB. BB \in\# CC \wedge AA = BB \cdot \sigma)$   
**unfolding** *subst\_cls\_mset\_def* **by** *auto*

### 7.3.5 Signs and Substitutions

**lemma** *subst\_lit\_is\_neg*[simp]:  $is\_neg (L \cdot l \sigma) = is\_neg L$   
**unfolding** *subst\_lit\_def* **by** *auto*

**lemma** *subst\_lit\_is\_pos*[simp]:  $is\_pos (L \cdot l \sigma) = is\_pos L$   
**unfolding** *subst\_lit\_def* **by** *auto*

**lemma** *subst\_minus*[simp]:  $(- L) \cdot l \mu = - (L \cdot l \mu)$   
**by** (*simp add: literal.map\_sel subst\_lit\_def uminus\_literal\_def*)

### 7.3.6 Substitution on Literal(s)

**lemma** *eqL\_neg\_lit\_eqL\_atm*[simp]:  $(Neg A' \cdot l \eta) = Neg A \longleftrightarrow A' \cdot a \eta = A$   
**by** (*simp add: subst\_lit\_def*)

**lemma** *eqL\_pos\_lit\_eqL\_atm*[simp]:  $(Pos A' \cdot l \eta) = Pos A \longleftrightarrow A' \cdot a \eta = A$   
**by** (*simp add: subst\_lit\_def*)

**lemma** *subst\_cls\_negs*[simp]:  $(negs AA) \cdot \sigma = negs (AA \cdot am \sigma)$   
**unfolding** *subst\_cls\_def* *subst\_lit\_def* *subst\_atm\_mset\_def* **by** *auto*

**lemma** *subst\_cls\_poss*[simp]:  $(poss AA) \cdot \sigma = poss (AA \cdot am \sigma)$   
**unfolding** *subst\_cls\_def* *subst\_lit\_def* *subst\_atm\_mset\_def* **by** *auto*

**lemma** *atms\_of\_subst\_atms*:  $atms\_of C \cdot as \sigma = atms\_of (C \cdot \sigma)$

**proof** –

**have**  $atms\_of (C \cdot \sigma) = set\_mset (image\_mset atm\_of (image\_mset (map\_literal (\lambda A. A \cdot a \sigma)) C))$   
**unfolding** *subst\_cls\_def* *subst\_atms\_def* *subst\_lit\_def* *atms\_of\_def* **by** *auto*  
**also have**  $\dots = set\_mset (image\_mset (\lambda A. A \cdot a \sigma) (image\_mset atm\_of C))$   
**by** *simp (meson literal.map\_sel)*  
**finally show**  $atms\_of C \cdot as \sigma = atms\_of (C \cdot \sigma)$   
**unfolding** *subst\_atms\_def* *atms\_of\_def* **by** *auto*

**qed**

**lemma** *in\_image\_Neg\_is\_neg*[simp]:  $L \cdot l \sigma \in Neg \text{ ‘ } AA \implies is\_neg L$   
**by** (*metis be\_x\_imageD literal.disc(2) literal.map\_disc\_iff subst\_lit\_def*)

**lemma** *subst\_lit\_in\_negs\_subst\_is\_neg*:  $L \cdot l \sigma \in\# (negs AA) \cdot \tau \implies is\_neg L$   
**by** *simp*

**lemma** *subst\_lit\_in\_negs\_is\_neg*:  $L \cdot l \sigma \in\# negs AA \implies is\_neg L$   
**by** *simp*

### 7.3.7 Substitution on Empty

**lemma** *subst\_atms\_empty*[simp]:  $\{\} \cdot as \sigma = \{\}$   
**unfolding** *subst\_atms\_def* **by** *auto*

**lemma** *subst\_atmss\_empty*[simp]:  $\{\} \cdot ass \sigma = \{\}$

**unfolding** *subst\_atmss\_def* **by** *auto*

**lemma** *comp\_substs\_empty\_iff[simp]*:  $\sigma s \odot s \eta s = [] \longleftrightarrow \sigma s = [] \vee \eta s = []$   
**using** *comp\_substs\_def map2\_empty\_iff* **by** *auto*

**lemma** *subst\_atm\_list\_empty[simp]*:  $[] \cdot al \sigma = []$   
**unfolding** *subst\_atm\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_empty[simp]*:  $\{\#\} \cdot am \sigma = \{\#\}$   
**unfolding** *subst\_atm\_mset\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_list\_empty[simp]*:  $[] \cdot aml \sigma = []$   
**unfolding** *subst\_atm\_mset\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_lists\_empty[simp]*:  $[] \cdot \cdot aml \sigma s = []$   
**unfolding** *subst\_atm\_mset\_lists\_def* **by** *auto*

**lemma** *subst\_cls\_empty[simp]*:  $\{\#\} \cdot \sigma = \{\#\}$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_empty[simp]*:  $\{\} \cdot cs \sigma = \{\}$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_list\_empty[simp]*:  $[] \cdot cl \sigma = []$   
**unfolding** *subst\_cls\_list\_def* **by** *auto*

**lemma** *subst\_cls\_lists\_empty[simp]*:  $[] \cdot \cdot cl \sigma s = []$   
**unfolding** *subst\_cls\_lists\_def* **by** *auto*

**lemma** *subst\_scls\_mset\_empty[simp]*:  $\{\#\} \cdot cm \sigma = \{\#\}$   
**unfolding** *subst\_cls\_mset\_def* **by** *auto*

**lemma** *subst\_atms\_empty\_iff[simp]*:  $AA \cdot as \eta = \{\} \longleftrightarrow AA = \{\}$   
**unfolding** *subst\_atms\_def* **by** *auto*

**lemma** *subst\_atmss\_empty\_iff[simp]*:  $AAA \cdot ass \eta = \{\} \longleftrightarrow AAA = \{\}$   
**unfolding** *subst\_atmss\_def* **by** *auto*

**lemma** *subst\_atm\_list\_empty\_iff[simp]*:  $As \cdot al \eta = [] \longleftrightarrow As = []$   
**unfolding** *subst\_atm\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_empty\_iff[simp]*:  $AA \cdot am \eta = \{\#\} \longleftrightarrow AA = \{\#\}$   
**unfolding** *subst\_atm\_mset\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_list\_empty\_iff[simp]*:  $AAs \cdot aml \eta = [] \longleftrightarrow AAs = []$   
**unfolding** *subst\_atm\_mset\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_lists\_empty\_iff[simp]*:  $AAs \cdot \cdot aml \eta s = [] \longleftrightarrow (AAs = [] \vee \eta s = [])$   
**using** *map2\_empty\_iff subst\_atm\_mset\_lists\_def* **by** *auto*

**lemma** *subst\_cls\_empty\_iff[simp]*:  $C \cdot \eta = \{\#\} \longleftrightarrow C = \{\#\}$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_empty\_iff[simp]*:  $CC \cdot cs \eta = \{\} \longleftrightarrow CC = \{\}$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_list\_empty\_iff[simp]*:  $Cs \cdot cl \eta = [] \longleftrightarrow Cs = []$   
**unfolding** *subst\_cls\_list\_def* **by** *auto*

**lemma** *subst\_cls\_lists\_empty\_iff[simp]*:  $Cs \cdot \cdot cl \eta s = [] \longleftrightarrow (Cs = [] \vee \eta s = [])$   
**using** *map2\_empty\_iff subst\_cls\_lists\_def* **by** *auto*

**lemma** *subst\_cls\_mset\_empty\_iff[simp]*:  $CC \cdot cm \eta = \{\#\} \longleftrightarrow CC = \{\#\}$

`unfolding subst_cls_mset_def by auto`

### 7.3.8 Substitution on a Union

`lemma subst_atms_union[simp]:  $(AA \cup BB) \cdot as \sigma = AA \cdot as \sigma \cup BB \cdot as \sigma$`   
`unfolding subst_atms_def by auto`

`lemma subst_atmss_union[simp]:  $(AAA \cup BBB) \cdot ass \sigma = AAA \cdot ass \sigma \cup BBB \cdot ass \sigma$`   
`unfolding subst_atmss_def by auto`

`lemma subst_atm_list_append[simp]:  $(As @ Bs) \cdot al \sigma = As \cdot al \sigma @ Bs \cdot al \sigma$`   
`unfolding subst_atm_list_def by auto`

`lemma subst_atm_mset_union[simp]:  $(AA + BB) \cdot am \sigma = AA \cdot am \sigma + BB \cdot am \sigma$`   
`unfolding subst_atm_mset_def by auto`

`lemma subst_atm_mset_list_append[simp]:  $(AAs @ BBs) \cdot aml \sigma = AAs \cdot aml \sigma @ BBs \cdot aml \sigma$`   
`unfolding subst_atm_mset_list_def by auto`

`lemma subst_cls_union[simp]:  $(C + D) \cdot \sigma = C \cdot \sigma + D \cdot \sigma$`   
`unfolding subst_cls_def by auto`

`lemma subst_cls_union[simp]:  $(CC \cup DD) \cdot cs \sigma = CC \cdot cs \sigma \cup DD \cdot cs \sigma$`   
`unfolding subst_cls_def by auto`

`lemma subst_cls_list_append[simp]:  $(Cs @ Ds) \cdot cl \sigma = Cs \cdot cl \sigma @ Ds \cdot cl \sigma$`   
`unfolding subst_cls_list_def by auto`

`lemma subst_cls_mset_union[simp]:  $(CC + DD) \cdot cm \sigma = CC \cdot cm \sigma + DD \cdot cm \sigma$`   
`unfolding subst_cls_mset_def by auto`

### 7.3.9 Substitution on a Singleton

`lemma subst_atms_single[simp]:  $\{A\} \cdot as \sigma = \{A \cdot a \sigma\}$`   
`unfolding subst_atms_def by auto`

`lemma subst_atmss_single[simp]:  $\{AA\} \cdot ass \sigma = \{AA \cdot as \sigma\}$`   
`unfolding subst_atmss_def by auto`

`lemma subst_atm_list_single[simp]:  $[A] \cdot al \sigma = [A \cdot a \sigma]$`   
`unfolding subst_atm_list_def by auto`

`lemma subst_atm_mset_single[simp]:  $\{\#A\# \} \cdot am \sigma = \{\#A \cdot a \sigma\# \}$`   
`unfolding subst_atm_mset_def by auto`

`lemma subst_atm_mset_list[simp]:  $[AA] \cdot aml \sigma = [AA \cdot am \sigma]$`   
`unfolding subst_atm_mset_list_def by auto`

`lemma subst_cls_single[simp]:  $\{\#L\# \} \cdot \sigma = \{\#L \cdot l \sigma\# \}$`   
`by simp`

`lemma subst_cls_single[simp]:  $\{C\} \cdot cs \sigma = \{C \cdot \sigma\}$`   
`unfolding subst_cls_def by auto`

`lemma subst_cls_list_single[simp]:  $[C] \cdot cl \sigma = [C \cdot \sigma]$`   
`unfolding subst_cls_list_def by auto`

`lemma subst_cls_mset_single[simp]:  $\{\#C\# \} \cdot cm \sigma = \{\#C \cdot \sigma\# \}$`   
`by simp`

### 7.3.10 Substitution on $op \#$

`lemma subst_atm_list.Cons[simp]:  $(A \# As) \cdot al \sigma = A \cdot a \sigma \# As \cdot al \sigma$`   
`unfolding subst_atm_list_def by auto`



**lemma** *subst\_atm\_mset\_list\_Cons*[simp]:  $(A \# As) \cdot aml \sigma = A \cdot am \sigma \# As \cdot aml \sigma$   
**unfolding** *subst\_atm\_mset\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_lists\_Cons*[simp]:  $(C \# Cs) \cdot aml (\sigma \# \sigma s) = C \cdot am \sigma \# Cs \cdot aml \sigma s$   
**unfolding** *subst\_atm\_mset\_lists\_def* **by** *auto*

**lemma** *subst\_cls\_list\_Cons*[simp]:  $(C \# Cs) \cdot cl \sigma = C \cdot \sigma \# Cs \cdot cl \sigma$   
**unfolding** *subst\_cls\_list\_def* **by** *auto*

**lemma** *subst\_cls\_lists\_Cons*[simp]:  $(C \# Cs) \cdot cl (\sigma \# \sigma s) = C \cdot \sigma \# Cs \cdot cl \sigma s$   
**unfolding** *subst\_cls\_lists\_def* **by** *auto*

### 7.3.11 Substitution on *tl*

**lemma** *subst\_atm\_list\_tl*[simp]:  $tl (As \cdot al \eta) = tl As \cdot al \eta$   
**by** (*induction As*) *auto*

**lemma** *subst\_atm\_mset\_list\_tl*[simp]:  $tl (AAs \cdot aml \eta) = tl AAs \cdot aml \eta$   
**by** (*induction AAs*) *auto*

### 7.3.12 Substitution on *op !*

**lemma** *comp\_substs\_nth*[simp]:  
 $length \tau s = length \sigma s \implies i < length \tau s \implies (\tau s \odot s \sigma s) ! i = (\tau s ! i) \odot (\sigma s ! i)$   
**by** (*simp add: comp\_substs\_def*)

**lemma** *subst\_atm\_list\_nth*[simp]:  $i < length As \implies (As \cdot al \tau) ! i = As ! i \cdot a \tau$   
**unfolding** *subst\_atm\_list\_def* **using** *less\_Suc\_eq\_0\_disj nth\_map* **by** *force*

**lemma** *subst\_atm\_mset\_list\_nth*[simp]:  $i < length AAs \implies (AAs \cdot aml \eta) ! i = (AAs ! i) \cdot am \eta$   
**unfolding** *subst\_atm\_mset\_list\_def* **by** *auto*

**lemma** *subst\_atm\_mset\_lists\_nth*[simp]:  
 $length AAs = length \sigma s \implies i < length AAs \implies (AAs \cdot aml \sigma s) ! i = (AAs ! i) \cdot am (\sigma s ! i)$   
**unfolding** *subst\_atm\_mset\_lists\_def* **by** *auto*

**lemma** *subst\_cls\_list\_nth*[simp]:  $i < length Cs \implies (Cs \cdot cl \tau) ! i = (Cs ! i) \cdot \tau$   
**unfolding** *subst\_cls\_list\_def* **using** *less\_Suc\_eq\_0\_disj nth\_map* **by** (*induction Cs*) *auto*

**lemma** *subst\_cls\_lists\_nth*[simp]:  
 $length Cs = length \sigma s \implies i < length Cs \implies (Cs \cdot cl \sigma s) ! i = (Cs ! i) \cdot (\sigma s ! i)$   
**unfolding** *subst\_cls\_lists\_def* **by** *auto*

### 7.3.13 Substitution on Various Other Functions

**lemma** *subst\_cls\_image*[simp]:  $image f X \cdot cs \sigma = \{f x \cdot \sigma \mid x. x \in X\}$   
**unfolding** *subst\_cls\_def* **by** *auto*

**lemma** *subst\_cls\_mset\_image\_mset*[simp]:  $image\_mset f X \cdot cm \sigma = \{\# f x \cdot \sigma. x \in \# X \#\}$   
**unfolding** *subst\_cls\_mset\_def* **by** *auto*

**lemma** *mset\_subst\_atm\_list\_subst\_atm\_mset*[simp]:  $mset (As \cdot al \sigma) = mset (As) \cdot am \sigma$   
**unfolding** *subst\_atm\_list\_def* *subst\_atm\_mset\_def* **by** *auto*

**lemma** *mset\_subst\_cls\_list\_subst\_cls\_mset*:  $mset (Cs \cdot cl \sigma) = (mset Cs) \cdot cm \sigma$   
**unfolding** *subst\_cls\_mset\_def* *subst\_cls\_list\_def* **by** *auto*

**lemma** *sum\_list\_subst\_cls\_list\_subst\_cls*[simp]:  $sum\_list (Cs \cdot cl \eta) = sum\_list Cs \cdot \eta$   
**unfolding** *subst\_cls\_list\_def* **by** (*induction Cs*) *auto*

**lemma** *set\_mset\_subst\_cls\_mset\_subst\_cls*:  $set\_mset (CC \cdot cm \mu) = (set\_mset CC) \cdot cs \mu$   
**by** (*simp add: subst\_cls\_mset\_def subst\_cls\_def*)

**lemma** *Neg\_Melem\_subst\_atm\_subst\_cls[simp]*:  $\text{Neg } A \in\# C \implies \text{Neg } (A \cdot a \sigma) \in\# C \cdot \sigma$   
**by** (*metis Melem\_subst\_cls eql\_neg\_lit\_eql\_atm*)

**lemma** *Pos\_Melem\_subst\_atm\_subst\_cls[simp]*:  $\text{Pos } A \in\# C \implies \text{Pos } (A \cdot a \sigma) \in\# C \cdot \sigma$   
**by** (*metis Melem\_subst\_cls eql\_pos\_lit\_eql\_atm*)

**lemma** *in\_atms\_of\_subst[simp]*:  $B \in \text{atms\_of } C \implies B \cdot a \sigma \in \text{atms\_of } (C \cdot \sigma)$   
**by** (*metis atms\_of\_subst\_atms image\_iff subst\_atms\_def*)

### 7.3.14 Renamings

**lemma** *is\_renaming\_id\_subst[simp]*: *is\_renaming id\_subst*  
**unfolding** *is\_renaming\_def* **by** *simp*

**lemma** *is\_renamingD*:  $\text{is\_renaming } \sigma \implies (\forall A1 A2. A1 \cdot a \sigma = A2 \cdot a \sigma \longleftrightarrow A1 = A2)$   
**by** (*metis is\_renaming\_def subst\_atm\_comp\_subst subst\_atm\_id\_subst*)

**lemma** *inv\_renaming\_cancel\_r[simp]*:  $\text{is\_renaming } r \implies r \odot \text{inv\_renaming } r = \text{id\_subst}$   
**unfolding** *inv\_renaming\_def is\_renaming\_def* **by** (*metis (mono\_tags) someI\_ex*)

**lemma** *inv\_renaming\_cancel\_r\_list[simp]*:  
 $\text{is\_renaming\_list } rs \implies rs \odot s \text{ map inv\_renaming } rs = \text{replicate (length } rs) \text{ id\_subst}$   
**unfolding** *is\_renaming\_list\_def* **by** (*induction rs (auto simp add: comp\_substs\_def)*)

**lemma** *Nil\_comp\_substs[simp]*:  $[] \odot s = []$   
**unfolding** *comp\_substs\_def* **by** *auto*

**lemma** *comp\_substs\_Nil[simp]*:  $s \odot [] = []$   
**unfolding** *comp\_substs\_def* **by** *auto*

**lemma** *is\_renaming\_idempotent\_id\_subst*:  $\text{is\_renaming } r \implies r \odot r = r \implies r = \text{id\_subst}$   
**by** (*metis comp\_subst\_assoc comp\_subst\_id\_subst inv\_renaming\_cancel\_r*)

**lemma** *is\_renaming\_left\_id\_subst\_right\_id\_subst*:  
 $\text{is\_renaming } r \implies s \odot r = \text{id\_subst} \implies r \odot s = \text{id\_subst}$   
**by** (*metis comp\_subst\_assoc comp\_subst\_id\_subst is\_renaming\_def*)

**lemma** *is\_renaming\_closure*:  $\text{is\_renaming } r1 \implies \text{is\_renaming } r2 \implies \text{is\_renaming } (r1 \odot r2)$   
**unfolding** *is\_renaming\_def* **by** (*metis comp\_subst\_assoc comp\_subst\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atm[simp]*:  $\text{is\_renaming } \varrho \implies A \cdot a \varrho \cdot a \text{inv\_renaming } \varrho = A$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atm\_comp\_subst subst\_atm\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atms[simp]*:  $\text{is\_renaming } \varrho \implies AA \cdot as \varrho \cdot as \text{inv\_renaming } \varrho = AA$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atms\_comp\_subst subst\_atms\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atmss[simp]*:  $\text{is\_renaming } \varrho \implies AAA \cdot ass \varrho \cdot ass \text{inv\_renaming } \varrho = AAA$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atmss\_comp\_subst subst\_atmss\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atm\_list[simp]*:  $\text{is\_renaming } \varrho \implies As \cdot al \varrho \cdot al \text{inv\_renaming } \varrho = As$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atm\_list\_comp\_subst subst\_atm\_list\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atm\_mset[simp]*:  $\text{is\_renaming } \varrho \implies AA \cdot am \varrho \cdot am \text{inv\_renaming } \varrho = AA$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atm\_mset\_comp\_subst subst\_atm\_mset\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_atm\_mset\_list[simp]*:  $\text{is\_renaming } \varrho \implies (AAs \cdot aml \varrho) \cdot aml \text{inv\_renaming } \varrho = AAs$   
**by** (*metis inv\_renaming\_cancel\_r subst\_atm\_mset\_list\_comp\_subst subst\_atm\_mset\_list\_id\_subst*)

**lemma** *is\_renaming\_list\_inv\_renaming\_cancel\_atm\_mset\_lists[simp]*:  
 $\text{length } AAs = \text{length } \varrho s \implies \text{is\_renaming\_list } \varrho s \implies AAs \cdot aml \varrho s \cdot aml \text{ map inv\_renaming } \varrho s = AAs$   
**by** (*metis inv\_renaming\_cancel\_r\_list subst\_atm\_mset\_lists\_comp\_substs subst\_atm\_mset\_lists\_id\_subst*)

**lemma** *is\_renaming\_inv\_renaming\_cancel\_lit[simp]*:  $\text{is\_renaming } \varrho \implies (L \cdot l \varrho) \cdot l \text{inv\_renaming } \varrho = L$

by (metis inv\_renaming\_cancel\_r subst\_lit\_comp\_subst subst\_lit\_id\_subst)

**lemma** is\_renaming\_inv\_renaming\_cancel\_cls[simp]: is\_renaming  $\varrho \implies C \cdot \varrho \cdot \text{inv\_renaming } \varrho = C$   
by (metis inv\_renaming\_cancel\_r subst\_cls\_comp\_subst subst\_cls\_id\_subst)

**lemma** is\_renaming\_inv\_renaming\_cancel\_cls[simp]: is\_renaming  $\varrho \implies CC \cdot cs \ \varrho \cdot cs \text{ inv\_renaming } \varrho = CC$   
by (metis inv\_renaming\_cancel\_r subst\_cls\_id\_subst subst\_clscomp\_subst)

**lemma** is\_renaming\_inv\_renaming\_cancel\_cls\_list[simp]: is\_renaming  $\varrho \implies Cs \cdot cl \ \varrho \cdot cl \text{ inv\_renaming } \varrho = Cs$   
by (metis inv\_renaming\_cancel\_r subst\_cls\_list\_comp\_subst subst\_cls\_list\_id\_subst)

**lemma** is\_renaming\_list\_inv\_renaming\_cancel\_cls\_list[simp]:  
length  $Cs = \text{length } \varrho s \implies \text{is\_renaming\_list } \varrho s \implies Cs \cdot cl \ \varrho s \cdot cl \text{ map inv\_renaming } \varrho s = Cs$   
by (metis inv\_renaming\_cancel\_r\_list subst\_cls\_lists\_comp\_substs subst\_cls\_lists\_id\_subst)

**lemma** is\_renaming\_inv\_renaming\_cancel\_cls\_mset[simp]: is\_renaming  $\varrho \implies CC \cdot cm \ \varrho \cdot cm \text{ inv\_renaming } \varrho = CC$   
by (metis inv\_renaming\_cancel\_r subst\_cls\_mset\_comp\_subst subst\_cls\_mset\_id\_subst)

### 7.3.15 Monotonicity

**lemma** subst\_cls\_mono: set\_mset  $C \subseteq \text{set\_mset } D \implies \text{set\_mset } (C \cdot \sigma) \subseteq \text{set\_mset } (D \cdot \sigma)$   
by force

**lemma** subst\_cls\_mono\_mset:  $C \subseteq\# D \implies C \cdot \sigma \subseteq\# D \cdot \sigma$   
unfolding subst\_cls\_def by (metis mset\_subset\_eq\_exists\_conv subst\_cls\_union)

**lemma** subst\_subset\_mono:  $D \subseteq\# C \implies D \cdot \sigma \subseteq\# C \cdot \sigma$   
unfolding subst\_cls\_def by (simp add: image\_mset\_subset\_mono)

### 7.3.16 Size after Substitution

**lemma** size\_subst[simp]: size  $(D \cdot \sigma) = \text{size } D$   
unfolding subst\_cls\_def by auto

**lemma** subst\_atm\_list\_length[simp]: length  $(As \cdot al \ \sigma) = \text{length } As$   
unfolding subst\_atm\_list\_def by auto

**lemma** length\_subst\_atm\_mset\_list[simp]: length  $(AAs \cdot aml \ \eta) = \text{length } AAs$   
unfolding subst\_atm\_mset\_list\_def by auto

**lemma** subst\_atm\_mset\_lists\_length[simp]: length  $(AAs \cdot aml \ \sigma s) = \min (\text{length } AAs) (\text{length } \sigma s)$   
unfolding subst\_atm\_mset\_lists\_def by auto

**lemma** subst\_cls\_list\_length[simp]: length  $(Cs \cdot cl \ \sigma) = \text{length } Cs$   
unfolding subst\_cls\_list\_def by auto

**lemma** comp\_substs\_length[simp]: length  $(\tau s \odot s \ \sigma s) = \min (\text{length } \tau s) (\text{length } \sigma s)$   
unfolding comp\_substs\_def by auto

**lemma** subst\_cls\_lists\_length[simp]: length  $(Cs \cdot cl \ \sigma s) = \min (\text{length } Cs) (\text{length } \sigma s)$   
unfolding subst\_cls\_lists\_def by auto

### 7.3.17 Variable Disjointness

**lemma** var\_disjoint\_clauses:  
assumes var\_disjoint  $Cs$   
shows  $\forall \sigma s. \text{length } \sigma s = \text{length } Cs \longrightarrow (\exists \tau. Cs \cdot cl \ \sigma s = Cs \cdot cl \ \tau)$   
**proof** clarify  
fix  $\sigma s :: 's \text{ list}$   
assume  $a: \text{length } \sigma s = \text{length } Cs$   
then obtain  $\tau$  where  $\forall i < \text{length } Cs. \forall S. S \subseteq\# Cs ! i \longrightarrow S \cdot \sigma s ! i = S \cdot \tau$   
using assms unfolding var\_disjoint\_def by blast  
then have  $\forall i < \text{length } Cs. (Cs ! i) \cdot \sigma s ! i = (Cs ! i) \cdot \tau$   
by auto

**then have**  $Cs \cdot cl \sigma s = Cs \cdot cl \tau$   
**using a by** (*simp add: nth\_equalityI*)  
**then show**  $\exists \tau. Cs \cdot cl \sigma s = Cs \cdot cl \tau$   
**by** *auto*  
**qed**

### 7.3.18 Ground Expressions and Substitutions

**lemma** *ex\_ground\_subst*:  $\exists \sigma. is\_ground\_subst \sigma$   
**using** *make\_ground\_subst*[*of*  $\{\#\}$ ]  
**by** (*simp add: is\_ground\_cls\_def*)

**lemma** *is\_ground\_cls\_list.Cons*[*simp*]:  
 $is\_ground\_cls\_list (C \# Cs) = (is\_ground\_cls C \wedge is\_ground\_cls\_list Cs)$   
**unfolding** *is\_ground\_cls\_list\_def* **by** *auto*

**Ground union lemma** *is\_ground\_atms\_union*[*simp*]:  $is\_ground\_atms (AA \cup BB) \longleftrightarrow is\_ground\_atms AA \wedge is\_ground\_atms BB$   
**unfolding** *is\_ground\_atms\_def* **by** *auto*

**lemma** *is\_ground\_atm\_mset\_union*[*simp*]:  
 $is\_ground\_atm\_mset (AA + BB) \longleftrightarrow is\_ground\_atm\_mset AA \wedge is\_ground\_atm\_mset BB$   
**unfolding** *is\_ground\_atm\_mset\_def* **by** *auto*

**lemma** *is\_ground\_cls\_union*[*simp*]:  $is\_ground\_cls (C + D) \longleftrightarrow is\_ground\_cls C \wedge is\_ground\_cls D$   
**unfolding** *is\_ground\_cls\_def* **by** *auto*

**lemma** *is\_ground\_clss\_union*[*simp*]:  
 $is\_ground\_clss (CC \cup DD) \longleftrightarrow is\_ground\_clss CC \wedge is\_ground\_clss DD$   
**unfolding** *is\_ground\_clss\_def* **by** *auto*

**lemma** *is\_ground\_cls\_list.is\_ground\_cls\_sum\_list*[*simp*]:  
 $is\_ground\_cls\_list Cs \implies is\_ground\_cls (sum\_list Cs)$   
**by** (*meson in\_mset\_sum\_list2 is\_ground\_cls\_def is\_ground\_cls\_list\_def*)

**Ground mono lemma** *is\_ground\_cls\_mono*:  $C \subseteq \# D \implies is\_ground\_cls D \implies is\_ground\_cls C$   
**unfolding** *is\_ground\_cls\_def* **by** (*metis set\_mset\_mono subsetD*)

**lemma** *is\_ground\_clss\_mono*:  $CC \subseteq DD \implies is\_ground\_clss DD \implies is\_ground\_clss CC$   
**unfolding** *is\_ground\_clss\_def* **by** *blast*

**lemma** *grounding\_of\_clss\_mono*:  $CC \subseteq DD \implies grounding\_of\_clss CC \subseteq grounding\_of\_clss DD$   
**using** *grounding\_of\_clss\_def* **by** *auto*

**lemma** *sum\_list\_subseteq\_mset\_is\_ground\_cls\_list*[*simp*]:  
 $sum\_list Cs \subseteq \# sum\_list Ds \implies is\_ground\_cls\_list Ds \implies is\_ground\_cls\_list Cs$   
**by** (*meson in\_mset\_sum\_list is\_ground\_cls\_def is\_ground\_cls\_list.is\_ground\_cls\_sum\_list is\_ground\_cls\_mono is\_ground\_cls\_list\_def*)

**Substituting on ground expression preserves ground lemma** *is\_ground\_comp\_subst*[*simp*]:  $is\_ground\_subst \sigma \implies is\_ground\_subst (\tau \odot \sigma)$   
**unfolding** *is\_ground\_subst\_def is\_ground\_atm\_def* **by** *auto*

**lemma** *ground\_subst\_ground\_atm*[*simp*]:  $is\_ground\_subst \sigma \implies is\_ground\_atm (A \cdot a \sigma)$   
**by** (*simp add: is\_ground\_subst\_def*)

**lemma** *ground\_subst\_ground\_lit*[*simp*]:  $is\_ground\_subst \sigma \implies is\_ground\_lit (L \cdot l \sigma)$   
**unfolding** *is\_ground\_lit\_def subst\_lit\_def* **by** (*cases L*) *auto*

**lemma** *ground\_subst\_ground\_cls*[*simp*]:  $is\_ground\_subst \sigma \implies is\_ground\_cls (C \cdot \sigma)$   
**unfolding** *is\_ground\_cls\_def* **by** *auto*

**lemma** *ground\_subst\_ground\_clss*[*simp*]:  $is\_ground\_subst \sigma \implies is\_ground\_clss (CC \cdot cs \sigma)$

**unfolding** *is\_ground\_clss\_def subst\_clss\_def* **by** *auto*

**lemma** *ground\_subst\_ground\_cls\_list[simp]*:  $\text{is\_ground\_subst } \sigma \implies \text{is\_ground\_cls\_list } (Cs \cdot cl \ \sigma)$   
**unfolding** *is\_ground\_cls\_list\_def subst\_cls\_list\_def* **by** *auto*

**lemma** *ground\_subst\_ground\_cls\_lists[simp]*:  
 $\forall \sigma \in \text{set } \sigma s. \text{is\_ground\_subst } \sigma \implies \text{is\_ground\_cls\_list } (Cs \cdot cl \ \sigma s)$   
**unfolding** *is\_ground\_cls\_list\_def subst\_cls\_lists\_def* **by** (*auto simp: set\_zip*)

**Substituting on ground expression has no effect** **lemma** *is\_ground\_subst\_atm[simp]*:  $\text{is\_ground\_atm } A \implies A \cdot a \ \sigma = A$   
**unfolding** *is\_ground\_atm\_def* **by** *simp*

**lemma** *is\_ground\_subst\_atms[simp]*:  $\text{is\_ground\_atms } AA \implies AA \cdot as \ \sigma = AA$   
**unfolding** *is\_ground\_atms\_def subst\_atms\_def image\_def* **by** *auto*

**lemma** *is\_ground\_subst\_atm\_mset[simp]*:  $\text{is\_ground\_atm\_mset } AA \implies AA \cdot am \ \sigma = AA$   
**unfolding** *is\_ground\_atm\_mset\_def subst\_atm\_mset\_def* **by** *auto*

**lemma** *is\_ground\_subst\_atm\_list[simp]*:  $\text{is\_ground\_atm\_list } As \implies As \cdot al \ \sigma = As$   
**unfolding** *is\_ground\_atm\_list\_def subst\_atm\_list\_def* **by** (*auto intro: nth\_equalityI*)

**lemma** *is\_ground\_subst\_atm\_list\_member[simp]*:  
 $\text{is\_ground\_atm\_list } As \implies i < \text{length } As \implies As ! i \cdot a \ \sigma = As ! i$   
**unfolding** *is\_ground\_atm\_list\_def* **by** *auto*

**lemma** *is\_ground\_subst\_lit[simp]*:  $\text{is\_ground\_lit } L \implies L \cdot l \ \sigma = L$   
**unfolding** *is\_ground\_lit\_def subst\_lit\_def* **by** (*cases L*) *simp\_all*

**lemma** *is\_ground\_subst\_cls[simp]*:  $\text{is\_ground\_cls } C \implies C \cdot \sigma = C$   
**unfolding** *is\_ground\_cls\_def subst\_cls\_def* **by** *simp*

**lemma** *is\_ground\_subst\_clss[simp]*:  $\text{is\_ground\_clss } CC \implies CC \cdot cs \ \sigma = CC$   
**unfolding** *is\_ground\_clss\_def subst\_clss\_def image\_def* **by** *auto*

**lemma** *is\_ground\_subst\_cls\_lists[simp]*:  
**assumes**  $\text{length } P = \text{length } Cs$  **and**  $\text{is\_ground\_cls\_list } Cs$   
**shows**  $Cs \cdot cl \ P = Cs$   
**using** *assms* **by** (*metis is\_ground\_cls\_list\_def is\_ground\_subst\_cls min.idem nth\_equalityI nth\_mem subst\_cls\_lists\_nth subst\_cls\_lists\_length*)

**lemma** *is\_ground\_subst\_lit\_iff*:  $\text{is\_ground\_lit } L \longleftrightarrow (\forall \sigma. L = L \cdot l \ \sigma)$   
**using** *is\_ground\_atm\_def is\_ground\_lit\_def subst\_lit\_def* **by** (*cases L*) *auto*

**lemma** *is\_ground\_subst\_cls\_iff*:  $\text{is\_ground\_cls } C \longleftrightarrow (\forall \sigma. C = C \cdot \sigma)$   
**by** (*metis ex\_ground\_subst ground\_subst\_ground\_cls is\_ground\_subst\_cls*)

**Members of ground expressions are ground** **lemma** *is\_ground\_cls\_as\_atms*:  $\text{is\_ground\_cls } C \longleftrightarrow (\forall A \in \text{atms\_of } C. \text{is\_ground\_atm } A)$   
**by** (*auto simp: atms\_of is\_ground\_cls\_def is\_ground\_lit\_def*)

**lemma** *is\_ground\_cls\_imp\_is\_ground\_lit*:  $L \in \# C \implies \text{is\_ground\_cls } C \implies \text{is\_ground\_lit } L$   
**by** (*simp add: is\_ground\_cls\_def*)

**lemma** *is\_ground\_cls\_imp\_is\_ground\_atm*:  $A \in \text{atms\_of } C \implies \text{is\_ground\_cls } C \implies \text{is\_ground\_atm } A$   
**by** (*simp add: is\_ground\_cls\_as\_atms*)

**lemma** *is\_ground\_cls\_is\_ground\_atms\_atms\_of[simp]*:  $\text{is\_ground\_cls } C \implies \text{is\_ground\_atms } (\text{atms\_of } C)$   
**by** (*simp add: is\_ground\_cls\_imp\_is\_ground\_atm is\_ground\_atms\_def*)

**lemma** *grounding\_ground*:  $C \in \text{grounding\_of\_clss } M \implies \text{is\_ground\_cls } C$   
**unfolding** *grounding\_of\_clss\_def grounding\_of\_cls\_def* **by** *auto*

**lemma** *in\_subset\_eq\_grounding\_of\_cls\_is\_ground\_cls*[simp]:  
 $C \in CC \implies CC \subseteq \text{grounding\_of\_cls } DD \implies \text{is\_ground\_cls } C$   
**unfolding** *grounding\_of\_cls\_def* *grounding\_of\_cls\_def* **by** *auto*

**lemma** *is\_ground\_cls\_empty*[simp]: *is\_ground\_cls*  $\{\#\}$   
**unfolding** *is\_ground\_cls\_def* **by** *simp*

**lemma** *grounding\_of\_cls\_ground*: *is\_ground\_cls*  $C \implies \text{grounding\_of\_cls } C = \{C\}$   
**unfolding** *grounding\_of\_cls\_def* **by** (*simp* *add*: *ex\_ground\_subst*)

**lemma** *grounding\_of\_cls\_empty*[simp]: *grounding\_of\_cls*  $\{\#\} = \{\{\#\}\}$   
**by** (*simp* *add*: *grounding\_of\_cls\_ground*)

### 7.3.19 Subsumption

**lemma** *subsumes\_empty\_left*[simp]: *subsumes*  $\{\#\}$   $C$   
**unfolding** *subsumes\_def* *subst\_cls\_def* **by** *simp*

**lemma** *strictly\_subsumes\_empty\_left*[simp]: *strictly\_subsumes*  $\{\#\}$   $C \longleftrightarrow C \neq \{\#\}$   
**unfolding** *strictly\_subsumes\_def* *subsumes\_def* *subst\_cls\_def* **by** *simp*

### 7.3.20 Unifiers

**lemma** *card\_le\_one\_alt*: *finite*  $X \implies \text{card } X \leq 1 \longleftrightarrow X = \{\}$   $\vee (\exists x. X = \{x\})$   
**by** (*induct* *rule*: *finite\_induct*) *auto*

**lemma** *is\_unifier\_subst\_atm\_eqI*:  
**assumes** *finite*  $AA$   
**shows** *is\_unifier*  $\sigma$   $AA \implies A \in AA \implies B \in AA \implies A \cdot a \sigma = B \cdot a \sigma$   
**unfolding** *is\_unifier\_def* *subst\_atms\_def* *card\_le\_one\_alt* [*OF* *finite\_imageI* [*OF* *assms*]]  
**by** (*metis* *equals0D* *imageI* *insert\_iff*)

**lemma** *is\_unifier\_alt*:  
**assumes** *finite*  $AA$   
**shows** *is\_unifier*  $\sigma$   $AA \longleftrightarrow (\forall A \in AA. \forall B \in AA. A \cdot a \sigma = B \cdot a \sigma)$   
**unfolding** *is\_unifier\_def* *subst\_atms\_def* *card\_le\_one\_alt* [*OF* *finite\_imageI* [*OF* *assms* (1)]]  
**by** (*rule* *iffI*, *metis* *empty\_iff* *insert\_iff* *insert\_image*, *blast*)

**lemma** *is\_unifiers\_subst\_atm\_eqI*:  
**assumes** *finite*  $AA$  *is\_unifiers*  $\sigma$   $AAA$   $AA \in AAA$   $A \in AA$   $B \in AA$   
**shows**  $A \cdot a \sigma = B \cdot a \sigma$   
**by** (*metis* *assms* *is\_unifiers\_def* *is\_unifier\_subst\_atm\_eqI*)

**theorem** *is\_unifiers\_comp*:  
 $\text{is\_unifiers } \sigma (\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ Bs) \cdot \text{ass } \eta) \longleftrightarrow$   
 $\text{is\_unifiers } (\eta \odot \sigma) (\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ Bs))$   
**unfolding** *is\_unifiers\_def* *is\_unifier\_def* *subst\_atms\_def* **by** *auto*

### 7.3.21 Most General Unifier

**lemma** *is\_mgu\_is\_unifiers*: *is\_mgu*  $\sigma$   $AAA \implies \text{is\_unifiers } \sigma$   $AAA$   
**using** *is\_mgu\_def* **by** *blast*

**lemma** *is\_mgu\_is\_most\_general*: *is\_mgu*  $\sigma$   $AAA \implies \text{is\_unifiers } \tau$   $AAA \implies \exists \gamma. \tau = \sigma \odot \gamma$   
**using** *is\_mgu\_def* **by** *blast*

**lemma** *is\_unifiers\_is\_unifier*: *is\_unifiers*  $\sigma$   $AAA \implies AA \in AAA \implies \text{is\_unifier } \sigma$   $AA$   
**using** *is\_unifiers\_def* **by** *simp*

### 7.3.22 Generalization and Subsumption

**lemma** *variants\_iff\_subsumes*: *variants*  $C$   $D \longleftrightarrow \text{subsumes } C$   $D \wedge \text{subsumes } D$   $C$

**proof**

**assume** *variants*  $C$   $D$

**then show** *subsumes*  $C$   $D \wedge \text{subsumes } D$   $C$

```

    unfolding variants_def generalizes_cls_def subsumes_def by (metis subset_mset.order.refl)
next
assume sub: subsumes C D ∧ subsumes D C
then have size C = size D
    unfolding subsumes_def by (metis antisym size_mset.mono size_subst)
then show variants C D
    using sub unfolding subsumes_def variants_def generalizes_cls_def
    by (metis leD mset_subset.size size_mset.mono size_subst
        subset_mset.order.not_eq_order.implies_strict)
qed

lemma wf_strictly_generalizes_cls: wfP strictly_generalizes_cls
proof -
{
    assume ∃ C_at. ∀ i. strictly_generalizes_cls (C_at (Suc i)) (C_at i)
    then obtain C_at :: nat ⇒ 'a clause where
        sg_C: ⋀ i. strictly_generalizes_cls (C_at (Suc i)) (C_at i)
        by blast

    define n :: nat where
        n = size (C_at 0)

    have sz_C: size (C_at i) = n for i
    proof (induct i)
        case (Suc i)
        then show ?case
            using sg_C[of i] unfolding strictly_generalizes_cls_def generalizes_cls_def subst_cls_def
            by (metis size_image_mset)
        qed (simp add: n_def)

    obtain σ_at :: nat ⇒ 's where
        C_σ: ⋀ i. image_mset (λL. L · l σ_at i) (C_at (Suc i)) = C_at i
        using sg_C[unfolded strictly_generalizes_cls_def generalizes_cls_def subst_cls_def] by metis

    define Ls_at :: nat ⇒ 'a literal list where
        Ls_at = rec_nat (SOME Ls. mset Ls = C_at 0)
        (λi Lsi. SOME Ls. mset Ls = C_at (Suc i) ∧ map (λL. L · l σ_at i) Ls = Lsi)

    have
        Ls_at_0: Ls_at 0 = (SOME Ls. mset Ls = C_at 0) and
        Ls_at_Suc: ⋀ i. Ls_at (Suc i) =
            (SOME Ls. mset Ls = C_at (Suc i) ∧ map (λL. L · l σ_at i) Ls = Ls_at i)
        unfolding Ls_at_def by simp+

    have mset_Lt_at_0: mset (Ls_at 0) = C_at 0
        unfolding Ls_at_0 by (rule someI_ex) (metis list_of_mset_ex)

    have mset (Ls_at (Suc i)) = C_at (Suc i) ∧ map (λL. L · l σ_at i) (Ls_at (Suc i)) = Ls_at i
        for i
    proof (induct i)
        case 0
        then show ?case
            by (simp add: Ls_at_Suc, rule someI_ex,
                metis C_σ image_mset_of_subset_list mset_Lt_at_0)
        next
        case Suc
        then show ?case
            by (subst (1 2) Ls_at_Suc) (rule someI_ex, metis C_σ image_mset_of_subset_list)
        qed
    note mset_Ls = this[THEN conjunct1] and Ls_σ = this[THEN conjunct2]

    have len_Ls: ⋀ i. length (Ls_at i) = n
        by (metis mset_Ls mset_Lt_at_0 not0_implies_Suc size_mset sz_C)

```

```

have is_pos_Ls:  $\bigwedge i j. j < n \implies is\_pos (Ls\_at (Suc i) ! j) \longleftrightarrow is\_pos (Ls\_at i ! j)$ 
  using Ls_σ len_Ls by (metis literal.map_disc_iff nth_map subst_lit_def)

have Ls_τ_strict_lit:  $\bigwedge i \tau. map (\lambda L. L \cdot l \tau) (Ls\_at i) \neq Ls\_at (Suc i)$ 
  by (metis C_σ mset_Ls Ls_σ mset_map sg_C generalizes_cls_def strictly_generalizes_cls_def
    subst_cls_def)

have Ls_τ_strict_tm:
  map (( $\lambda t. t \cdot a \tau$ )  $\circ$  atm_of) (Ls_at i)  $\neq$  map atm_of (Ls_at (Suc i)) for i  $\tau$ 
proof -
  obtain j :: nat where
    j_lt:  $j < n$  and
    j_τ:  $Ls\_at i ! j \cdot l \tau \neq Ls\_at (Suc i) ! j$ 
    using Ls_τ_strict_lit[of  $\tau$  i] len_Ls
    by (metis (no_types, lifting) length_map list_eq_iff_nth_eq nth_map)

  have atm_of (Ls_at i ! j)  $\cdot a \tau \neq$  atm_of (Ls_at (Suc i) ! j)
    using j_τ is_pos_Ls[OF j_lt]
    by (metis (mono_guards) literal.expand literal.map_disc_iff literal.map_sel subst_lit_def)
  then show ?thesis
    using j_lt len_Ls by (metis nth_map o_apply)
qed

define tm_at :: nat  $\Rightarrow$  'a where
   $\bigwedge i. tm\_at i = atm\_of\_atms (map atm\_of (Ls\_at i))$ 

have  $\bigwedge i. generalizes\_atm (tm\_at (Suc i)) (tm\_at i)$ 
  unfolding tm_at_def generalizes_atm_def atm_of_atms_subst
  using Ls_σ[THEN arg_cong, of map atm_of] by (auto simp: comp_def)
moreover have  $\bigwedge i. \neg generalizes\_atm (tm\_at i) (tm\_at (Suc i))$ 
  unfolding tm_at_def generalizes_atm_def atm_of_atms_subst by (simp add: Ls_τ_strict_tm)
ultimately have  $\bigwedge i. strictly\_generalizes\_atm (tm\_at (Suc i)) (tm\_at i)$ 
  unfolding strictly_generalizes_atm_def by blast
then have False
  using wf_strictly_generalizes_atm[unfolded wfP_def wf_iff_no_infinite_down_chain] by blast
}
then show wfP (strictly_generalizes_cls :: 'a clause  $\Rightarrow$  _  $\Rightarrow$  _)
  unfolding wfP_def by (blast intro: wf_iff_no_infinite_down_chain[THEN iffD2])
qed

lemma strict_subset_subst_strictly_subsumes:
  assumes cη_sub:  $C \cdot \eta \subset\# D$ 
  shows strictly_subsumes C D
  by (metis cη_sub leD mset_subset_size size_mset_mono size_subst strictly_subsumes_def
    subset_mset.dual_order.strict_implies_order substitution_ops.subsumes_def)

lemma subsumes_trans:  $subsumes C D \implies subsumes D E \implies subsumes C E$ 
  unfolding subsumes_def
  by (metis (no_types) subset_mset.order.trans subst_cls_comp_subst subst_cls_mono_mset)

lemma subset_strictly_subsumes:  $C \subset\# D \implies strictly\_subsumes C D$ 
  using strict_subset_subst_strictly_subsumes[of C id_subst] by auto

lemma strictly_subsumes_neq:  $strictly\_subsumes D' D \implies D' \neq D \cdot \sigma$ 
  unfolding strictly_subsumes_def subsumes_def by blast

lemma strictly_subsumes_has_minimum:
  assumes CC ≠ {}
  shows  $\exists C \in CC. \forall D \in CC. \neg strictly\_subsumes D C$ 
proof (rule ccontr)
  assume  $\neg (\exists C \in CC. \forall D \in CC. \neg strictly\_subsumes D C)$ 
  then have  $\forall C \in CC. \exists D \in CC. strictly\_subsumes D C$ 

```



```

  by blast
then obtain f where
  f_p:  $\forall C \in CC. f\ C \in CC \wedge \text{strictly\_subsumes}\ (f\ C)\ C$ 
  by metis
from assms obtain C where
  C_p:  $C \in CC$ 
  by auto

define c :: nat  $\Rightarrow$  'a clause where
   $\bigwedge n. c\ n = (f\ \wedge^n)\ C$ 

have incc:  $c\ i \in CC$  for i
  by (induction i) (auto simp: c_def f_p C_p)
have ps:  $\forall i. \text{strictly\_subsumes}\ (c\ (\text{Suc}\ i))\ (c\ i)$ 
  using incc f_p unfolding c_def by auto
have  $\forall i. \text{size}\ (c\ i) \geq \text{size}\ (c\ (\text{Suc}\ i))$ 
  using ps unfolding strictly_subsumes_def subsumes_def by (metis size_mset_mono size_subst)
then have lte:  $\forall i. (\text{size} \circ c)\ i \geq (\text{size} \circ c)\ (\text{Suc}\ i)$ 
  unfolding comp_def .
then have  $\exists l. \forall l' \geq l. \text{size}\ (c\ l') = \text{size}\ (c\ (\text{Suc}\ l'))$ 
  using f_Suc_decr_eventually_const comp_def by auto
then obtain l where
  l_p:  $\forall l' \geq l. \text{size}\ (c\ l') = \text{size}\ (c\ (\text{Suc}\ l'))$ 
  by metis
then have  $\forall l' \geq l. \text{strictly\_generalizes\_cls}\ (c\ (\text{Suc}\ l'))\ (c\ l')$ 
  using ps unfolding strictly_generalizes_cls_def generalizes_cls_def
  by (metis size_subst less_irrefl strictly_subsumes_def mset_subset_size
    subset_mset_def subsumes_def strictly_subsumes_neq)
then have  $\forall i. \text{strictly\_generalizes\_cls}\ (c\ (\text{Suc}\ i + l))\ (c\ (i + l))$ 
  unfolding strictly_generalizes_cls_def generalizes_cls_def by auto
then have  $\exists f. \forall i. \text{strictly\_generalizes\_cls}\ (f\ (\text{Suc}\ i))\ (f\ i)$ 
  by (rule exI[of  $\lambda x. c\ (x + l)$ ])
then show False
  using wf_strictly_generalizes_cls
  wf_iff_no_infinite_down_chain[of  $\{(x, y). \text{strictly\_generalizes\_cls}\ x\ y\}$ ]
  unfolding wfP_def by auto
qed

end

```

## 7.4 Most General Unifiers

```

locale mgu = substitution subst_atm id_subst comp_subst atm_of_atms renamings_apart
for
  subst_atm :: 'a  $\Rightarrow$  's  $\Rightarrow$  'a and
  id_subst :: 's and
  comp_subst :: 's  $\Rightarrow$  's  $\Rightarrow$  's and
  atm_of_atms :: 'a list  $\Rightarrow$  'a and
  renamings_apart :: 'a literal multiset list  $\Rightarrow$  's list +
fixes
  mgu :: 'a set set  $\Rightarrow$  's option
assumes
  mgu_sound:  $\text{finite}\ AAA \implies (\forall AA \in AAA. \text{finite}\ AA) \implies \text{mgu}\ AAA = \text{Some}\ \sigma \implies \text{is\_mgu}\ \sigma\ AAA$  and
  mgu_complete:
     $\text{finite}\ AAA \implies (\forall AA \in AAA. \text{finite}\ AA) \implies \text{is\_unifiers}\ \sigma\ AAA \implies \exists \tau. \text{mgu}\ AAA = \text{Some}\ \tau$ 
begin

lemmas is_unifiers_mgu = mgu_sound[unfolded is_mgu_def, THEN conjunct1]
lemmas is_mgu_most_general = mgu_sound[unfolded is_mgu_def, THEN conjunct2]

lemma mgu_unifier:
  assumes
    aslen:  $\text{length}\ As = n$  and
    aaslen:  $\text{length}\ AAs = n$  and

```

```

    mgu: Some  $\sigma = \text{mgu } (\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ AAs))$  and
    i_lt:  $i < n$  and
    a_in:  $A \in \# \ AAs ! i$ 
  shows  $A \cdot a \ \sigma = As ! i \cdot a \ \sigma$ 
proof -
  from mgu have is_mgu  $\sigma$  ( $\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ AAs)$ )
    using mgu_sound by auto
  then have is_unifiers  $\sigma$  ( $\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ AAs)$ )
    using is_mgu_is_unifiers by auto
  then have is_unifier  $\sigma$  ( $\text{set\_mset } (\text{add\_mset } (As ! i) (AAs ! i))$ )
    using i_lt aslen aaslen unfolding is_unifiers_def is_unifier_def
    by simp (metis length_zip min.idem nth_mem nth_zip prod.case set_mset_add_mset_insert)
  then show ?thesis
    using aslen aaslen a_in is_unifier_subst_atm_eqI
    by (metis finite_set_mset insertCI set_mset_add_mset_insert)
qed

end

end

```

## 8 Refutational Inference Systems

```

theory Inference_System
  imports Herbrand_Interpretation
begin

```

This theory gathers results from Section 2.4 (“Refutational Theorem Proving”), 3 (“Standard Resolution”), and 4.2 (“Counterexample-Reducing Inference Systems”) of Bachmair and Ganzinger’s chapter.

### 8.1 Preliminaries

Inferences have one distinguished main premise, any number of side premises, and a conclusion.

```

datatype 'a inference =
  Infer (side_prem_of: 'a clause multiset) (main_prem_of: 'a clause) (concl_of: 'a clause)

```

```

abbreviation prem_of :: 'a inference  $\Rightarrow$  'a clause multiset where
  prem_of  $\gamma \equiv \text{side\_prem\_of } \gamma + \{\# \text{main\_prem\_of } \gamma\}$ 

```

```

abbreviation concls_of :: 'a inference set  $\Rightarrow$  'a clause set where
  concls_of  $\Gamma \equiv \text{concl\_of } ' \Gamma$ 

```

```

definition infer_from :: 'a clause set  $\Rightarrow$  'a inference  $\Rightarrow$  bool where
  infer_from  $CC \ \gamma \longleftrightarrow \text{set\_mset } (\text{prem\_of } \gamma) \subseteq CC$ 

```

```

locale inference_system =
  fixes  $\Gamma :: 'a \text{ inference set}$ 
begin

```

```

definition inferences_from :: 'a clause set  $\Rightarrow$  'a inference set where
  inferences_from  $CC = \{\gamma. \gamma \in \Gamma \wedge \text{infer\_from } CC \ \gamma\}$ 

```

```

definition inferences_between :: 'a clause set  $\Rightarrow$  'a clause  $\Rightarrow$  'a inference set where
  inferences_between  $CC \ C = \{\gamma. \gamma \in \Gamma \wedge \text{infer\_from } (CC \cup \{C\}) \ \gamma \wedge C \in \# \ \text{prem\_of } \gamma\}$ 

```

```

lemma inferences_from_mono:  $CC \subseteq DD \Longrightarrow \text{inferences\_from } CC \subseteq \text{inferences\_from } DD$ 
  unfolding inferences_from_def infer_from_def by fast

```

```

definition saturated :: 'a clause set  $\Rightarrow$  bool where
  saturated  $N \longleftrightarrow \text{concls\_of } (\text{inferences\_from } N) \subseteq N$ 

```

```

lemma saturatedD:
  assumes
    satur: saturated N and
    inf: Infer CC D E  $\in \Gamma$  and
    cc_subs_n: set_mset CC  $\subseteq$  N and
    d_in_n: D  $\in$  N
  shows E  $\in$  N
proof –
  have Infer CC D E  $\in$  inferences_from N
  unfolding inferences_from_def infer_from_def using inf cc_subs_n d_in_n by simp
  then have E  $\in$  concls_of (inferences_from N)
  unfolding image_iff by (metis inference.sel(3))
  then show E  $\in$  N
  using satur unfolding saturated_def by blast
qed

end

Satisfiability preservation is a weaker requirement than soundness.

locale sat_preserving_inference_system = inference_system +
  assumes  $\Gamma_{\text{sat\_preserving}}$ : satisfiable N  $\implies$  satisfiable (N  $\cup$  concls_of (inferences_from N))

locale sound_inference_system = inference_system +
  assumes  $\Gamma_{\text{sound}}$ : Infer CC D E  $\in \Gamma \implies I \models_m CC \implies I \models D \implies I \models E$ 
begin

lemma  $\Gamma_{\text{sat\_preserving}}$ :
  assumes sat_n: satisfiable N
  shows satisfiable (N  $\cup$  concls_of (inferences_from N))
proof –
  obtain I where i: I  $\models_s$  N
  using sat_n by blast
  then have  $\bigwedge CC\ D\ E. \text{Infer } CC\ D\ E \in \Gamma \implies \text{set\_mset } CC \subseteq N \implies D \in N \implies I \models E$ 
  using  $\Gamma_{\text{sound}}$  unfolding true_cls_def true_cls_mset_def by (simp add: subset_eq)
  then have  $\bigwedge \gamma. \gamma \in \Gamma \implies \text{infer\_from } N\ \gamma \implies I \models \text{concl\_of } \gamma$ 
  unfolding infer_from_def by (case_tac  $\gamma$ ) clarsimp
  then have I  $\models_s$  concls_of (inferences_from N)
  unfolding inferences_from_def image_def true_cls_def infer_from_def by blast
  then have I  $\models_s$  N  $\cup$  concls_of (inferences_from N)
  using i by simp
  then show ?thesis
  by blast
qed

sublocale sat_preserving_inference_system
by unfold_locales (erule  $\Gamma_{\text{sat\_preserving}}$ )

end

locale reductive_inference_system = inference_system  $\Gamma$  for  $\Gamma :: ('a :: \text{wellorder}) \text{ inference set} +$ 
  assumes  $\Gamma_{\text{reductive}}$ :  $\gamma \in \Gamma \implies \text{concl\_of } \gamma < \text{main\_prem\_of } \gamma$ 

```

## 8.2 Refutational Completeness

Refutational completeness can be established once and for all for counterexample-reducing inference systems. The material formalized here draws from both the general framework of Section 4.2 and the concrete instances of Section 3.

```

locale counterex_reducing_inference_system =
  inference_system  $\Gamma$  for  $\Gamma :: ('a :: \text{wellorder}) \text{ inference set} +$ 
  fixes L_of ::  $'a \text{ clause set} \Rightarrow 'a \text{ interp}$ 
  assumes  $\Gamma_{\text{counterex\_reducing}}$ :
     $\{\#\} \notin N \implies D \in N \implies \neg \text{L\_of } N \models D \implies (\bigwedge C. C \in N \implies \neg \text{L\_of } N \models C \implies D \leq C) \implies$ 
     $\exists CC\ E. \text{set\_mset } CC \subseteq N \wedge \text{L\_of } N \models_m CC \wedge \text{Infer } CC\ D\ E \in \Gamma \wedge \neg \text{L\_of } N \models E \wedge E < D$ 

```

```

begin

lemma ex_min_counterex:
  fixes  $N :: ('a :: wellorder) \text{ clause set}$ 
  assumes  $\neg I \models N$ 
  shows  $\exists C \in N. \neg I \models C \wedge (\forall D \in N. D < C \longrightarrow I \models D)$ 
proof -
  obtain  $C$  where  $C \in N$  and  $\neg I \models C$ 
    using assms unfolding true_cls_def by auto
  then have  $c\_in: C \in \{C \in N. \neg I \models C\}$ 
    by blast
  show ?thesis
    using wf_eq_minimal[THEN iffD1, rule_format, OF wf_less_multiset c_in] by blast
qed

```

```

theorem saturated_model:
  assumes
    satur: saturated  $N$  and
    ec_ni_n:  $\{\#\} \notin N$ 
  shows  $L\_of\ N \models N$ 
proof -
  have ec_ni_n:  $\{\#\} \notin N$ 
    using ec_ni_n by auto

  {
    assume  $\neg L\_of\ N \models N$ 
    then obtain  $D$  where
      d_in_n:  $D \in N$  and
      d_cex:  $\neg L\_of\ N \models D$  and
      d_min:  $\bigwedge C. C \in N \implies C < D \implies L\_of\ N \models C$ 
      by (meson ex_min_counterex)
    then obtain  $CC\ E$  where
      cc_subs_n:  $set\_mset\ CC \subseteq N$  and
      inf_e:  $Infer\ CC\ D\ E \in \Gamma$  and
      e_cex:  $\neg L\_of\ N \models E$  and
      e_lt_d:  $E < D$ 
      using  $\Gamma\_counterex\_reducing[OF\ ec\_ni\_n]$  not_less by metis
    from cc_subs_n inf_e have  $E \in N$ 
      using d_in_n satur by (blast dest: saturatedD)
    then have False
      using e_cex e_lt_d d_min not_less by blast
  }
  then show ?thesis
    by satx
qed

```

Cf. Corollary 3.10:

```

corollary saturated_complete:  $saturated\ N \implies \neg\ satisfiable\ N \implies \{\#\} \in N$ 
  using saturated_model by blast

```

end

### 8.3 Compactness

Bachmair and Ganzinger claim that compactness follows from refutational completeness but leave the proof to the readers' imagination. Our proof relies on an inductive definition of saturation in terms of a base set of clauses.

```

context inference_system
begin

```

```

inductive-set saturate :: 'a clause set  $\Rightarrow$  'a clause set' for  $CC :: '$ a clause set' where

```

```

base:  $C \in CC \implies C \in \text{saturate } CC$ 
| step:  $\text{Infer } CC' D E \in \Gamma \implies (\bigwedge C'. C' \in \# CC' \implies C' \in \text{saturate } CC) \implies D \in \text{saturate } CC \implies E \in \text{saturate } CC$ 

```

```

lemma saturate_mono:  $C \in \text{saturate } CC \implies CC \subseteq DD \implies C \in \text{saturate } DD$ 
by (induct rule: saturate.induct) (auto intro: saturate.intros)

```

```

lemma saturated_saturation[simp, intro]: saturated (saturation N)
unfolding saturated_def inferences_from_def infer_from_def image_def
by clarify (rename_tac x, case_tac x, auto elim!: saturate.step)

```

```

lemma saturate_finite:  $C \in \text{saturation } CC \implies \exists DD. DD \subseteq CC \wedge \text{finite } DD \wedge C \in \text{saturation } DD$ 

```

```

proof (induct rule: saturate.induct)
case (base C)
then have  $\{C\} \subseteq CC$  and finite  $\{C\}$  and  $C \in \text{saturation } \{C\}$ 
by (auto intro: saturate.intros)
then show ?case
by blast
next
case (step CC' D E)
obtain DD_of where
 $\bigwedge C. C \in \# CC' \implies DD\_of C \subseteq CC \wedge \text{finite } (DD\_of C) \wedge C \in \text{saturation } (DD\_of C)$ 
using step(3) by metis
then have
 $(\bigcup C \in \text{set\_mset } CC'. DD\_of C) \subseteq CC$ 
finite  $(\bigcup C \in \text{set\_mset } CC'. DD\_of C) \wedge \text{set\_mset } CC' \subseteq \text{saturation } (\bigcup C \in \text{set\_mset } CC'. DD\_of C)$ 
by (auto intro: saturate_mono)
then obtain DD where
 $d\_sub: DD \subseteq CC$  and  $d\_fin: \text{finite } DD$  and  $in\_sat\_d: \text{set\_mset } CC' \subseteq \text{saturation } DD$ 
by blast
obtain EE where
 $e\_sub: EE \subseteq CC$  and  $e\_fin: \text{finite } EE$  and  $in\_sat\_ee: D \in \text{saturation } EE$ 
using step(5) by blast
have  $DD \cup EE \subseteq CC$ 
using d_sub e_sub step(1) by fast
moreover have finite  $(DD \cup EE)$ 
using d_fin e_fin by fast
moreover have  $E \in \text{saturation } (DD \cup EE)$ 
using in_sat_d in_sat_ee step.hyps(1)
by (blast intro: inference_system.saturation.step saturation_mono)
ultimately show ?case
by blast
qed

```

end

```

context sound_inference_system
begin

```

```

theorem saturation_sound:  $C \in \text{saturation } CC \implies I \models_s CC \implies I \models C$ 
by (induct rule: saturate.induct) (auto simp: true_cls_mset_def true_cls_def  $\Gamma\_sound$ )

```

end

```

context sat_preserving_inference_system
begin

```

This result surely holds, but we have yet to prove it. The challenge is: Every time a new clause is introduced, we also get a new interpretation (by the definition of *sat\_preserving\_inference\_system*). But the interpretation we want here is then the one that exists "at the limit". Maybe we can use compactness to prove it.

```

theorem saturation_sat_preserving: satisfiable  $CC \implies \text{satisfiable } (\text{saturation } CC)$ 
oops

```

end

**locale** *sound\_counterex\_reducing\_inference\_system* =  
   *counterex\_reducing\_inference\_system* + *sound\_inference\_system*  
**begin**

Compactness of clausal logic is stated as Theorem 3.12 for the case of unordered ground resolution. The proof below is a generalization to any sound counterexample-reducing inference system. The actual theorem will become available once the locale has been instantiated with a concrete inference system.

**theorem** *clausal\_logic\_compact*:  
   **fixes**  $N :: ('a :: \text{wellorder}) \text{ clause set}$   
   **shows**  $\neg \text{satisfiable } N \longleftrightarrow (\exists DD \subseteq N. \text{finite } DD \wedge \neg \text{satisfiable } DD)$   
**proof**  
   **assume**  $\neg \text{satisfiable } N$   
   **then have**  $\{\#\} \in \text{saturate } N$   
     **using** *saturated\_complete saturated\_saturate saturate.base unfolding true\_cls\_def by meson*  
   **then have**  $\exists DD \subseteq N. \text{finite } DD \wedge \{\#\} \in \text{saturate } DD$   
     **using** *saturate\_finite by fastforce*  
   **then show**  $\exists DD \subseteq N. \text{finite } DD \wedge \neg \text{satisfiable } DD$   
     **using** *saturate\_sound by auto*  
**next**  
   **assume**  $\exists DD \subseteq N. \text{finite } DD \wedge \neg \text{satisfiable } DD$   
   **then show**  $\neg \text{satisfiable } N$   
     **by** (*blast intro: true\_cls\_mono*)  
**qed**  
end  
end

## 9 Candidate Models for Ground Resolution

**theory** *Ground\_Resolution\_Model*  
**imports** *Herbrand\_Interpretation*  
**begin**

The proofs of refutational completeness for the two resolution inference systems presented in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter share mostly the same candidate model construction. The literal selection capability needed for the second system is ignored by the first one, by taking  $\lambda_. \{\}$  as instantiation for the  $S$  parameter.

**locale** *selection* =  
   **fixes**  $S :: 'a \text{ clause} \Rightarrow 'a \text{ clause}$   
   **assumes**  
     *S\_selects\_subseteq*:  $S \ C \subseteq_{\#} C$  **and**  
     *S\_selects\_neg\_lits*:  $L \in_{\#} S \ C \Longrightarrow \text{is\_neg } L$   
**locale** *ground\_resolution\_with\_selection* = *selection S*  
   **for**  $S :: ('a :: \text{wellorder}) \text{ clause} \Rightarrow 'a \text{ clause}$   
**begin**

The following commands corresponds to Definition 3.14, which generalizes Definition 3.1. *production C* is denoted  $\varepsilon_C$  in the chapter; *interp C* is denoted  $I_C$ ; *Interp C* is denoted  $I^C$ ; and *Interp\_N* is denoted  $I_N$ . The mutually recursive definition from the chapter is massaged to simplify the termination argument. The *production\_unfold* lemma below gives the intended characterization.

**context**  
   **fixes**  $N :: 'a \text{ clause set}$   
**begin**

**function** *production* ::  $'a \text{ clause} \Rightarrow 'a \text{ interp}$  **where**  
   *production C* =  
    $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max\_mset } C = \text{Pos } A \wedge \neg (\bigcup D \in \{D. D < C\}. \text{production } D) \models C \wedge S \ C = \{\#\}\}$

**by** *auto*  
**termination** **by** (*rule termination[OF wf, simplified]*)

**declare** *production.simps* [*simp del*]

**definition** *interp* :: '*a clause*  $\Rightarrow$  '*a interp* **where**  
*interp* *C* = ( $\bigcup D \in \{D. D < C\}. \text{production } D$ )

**lemma** *production\_unfold*:  
*production* *C* =  $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max\_mset } C = \text{Pos } A \wedge \neg \text{interp } C \models C \wedge S \ C = \{\#\}\}$   
**unfolding** *interp\_def* **by** (*rule production.simps*)

**abbreviation** *productive* :: '*a clause*  $\Rightarrow$  *bool* **where**  
*productive* *C*  $\equiv$  *production* *C*  $\neq \{\}$

**abbreviation** *produces* :: '*a clause*  $\Rightarrow$  '*a*  $\Rightarrow$  *bool* **where**  
*produces* *C* *A*  $\equiv$  *production* *C* =  $\{A\}$

**lemma** *producesD*: *produces* *C* *A*  $\Longrightarrow C \in N \wedge C \neq \{\#\} \wedge \text{Pos } A = \text{Max\_mset } C \wedge \neg \text{interp } C \models C \wedge S \ C = \{\#\}$   
**unfolding** *production\_unfold* **by** *auto*

**definition** *Interp* :: '*a clause*  $\Rightarrow$  '*a interp* **where**  
*Interp* *C* = *interp* *C*  $\cup$  *production* *C*

**lemma** *interp\_subseteq\_Interp*[*simp*]: *interp* *C*  $\subseteq$  *Interp* *C*  
**by** (*simp add: Interp\_def*)

**lemma** *Interp\_as\_UNION*: *Interp* *C* = ( $\bigcup D \in \{D. D \leq C\}. \text{production } D$ )  
**unfolding** *Interp\_def* *interp\_def* *less\_eq\_multiset\_def* **by** *fast*

**lemma** *productive\_not\_empty*: *productive* *C*  $\Longrightarrow C \neq \{\#\}$   
**unfolding** *production\_unfold* **by** *simp*

**lemma** *productive\_imp\_produces\_Max\_literal*: *productive* *C*  $\Longrightarrow$  *produces* *C* (*atm\_of* (*Max\_mset* *C*))  
**unfolding** *production\_unfold* **by** (*auto simp del: atm\_of\_Max\_lit*)

**lemma** *productive\_imp\_produces\_Max\_atom*: *productive* *C*  $\Longrightarrow$  *produces* *C* (*Max* (*atms\_of* *C*))  
**unfolding** *atms\_of\_def* *Max\_atm\_of\_set\_mset\_commute*[*OF productive\_not\_empty*]  
**by** (*rule productive\_imp\_produces\_Max\_literal*)

**lemma** *produces\_imp\_Max\_literal*: *produces* *C* *A*  $\Longrightarrow A = \text{atm\_of } (\text{Max\_mset } C)$   
**using** *productive\_imp\_produces\_Max\_literal* **by** *auto*

**lemma** *produces\_imp\_Max\_atom*: *produces* *C* *A*  $\Longrightarrow A = \text{Max } (\text{atms\_of } C)$   
**using** *producesD* *produces\_imp\_Max\_literal* **by** *auto*

**lemma** *produces\_imp\_Pos\_in\_lits*: *produces* *C* *A*  $\Longrightarrow \text{Pos } A \in \# \ C$   
**by** (*simp add: producesD*)

**lemma** *productive\_in\_N*: *productive* *C*  $\Longrightarrow C \in N$   
**unfolding** *production\_unfold* **by** *simp*

**lemma** *produces\_imp\_atms\_leq*: *produces* *C* *A*  $\Longrightarrow B \in \text{atms\_of } C \Longrightarrow B \leq A$   
**using** *Max.coboundedI* *produces\_imp\_Max\_atom* **by** *blast*

**lemma** *produces\_imp\_neg\_notin\_lits*: *produces* *C* *A*  $\Longrightarrow \neg \text{Neg } A \in \# \ C$   
**by** (*simp add: pos\_Max\_imp\_neg\_notin producesD*)

**lemma** *less\_eq\_imp\_interp\_subseteq\_interp*: *C*  $\leq D \Longrightarrow \text{interp } C \subseteq \text{interp } D$   
**unfolding** *interp\_def* **by** *auto* (*metis order.strict\_trans2*)

**lemma** *less\_eq\_imp\_interp\_subseteq\_Interp*: *C*  $\leq D \Longrightarrow \text{interp } C \subseteq \text{Interp } D$   
**unfolding** *Interp\_def* **using** *less\_eq\_imp\_interp\_subseteq\_interp* **by** *blast*

**lemma** *less\_imp\_production\_subseteq\_interp*:  $C < D \implies \text{production } C \subseteq \text{interp } D$   
**unfolding** *interp\_def* **by** *fast*

**lemma** *less\_eq\_imp\_production\_subseteq\_Interp*:  $C \leq D \implies \text{production } C \subseteq \text{Interp } D$   
**unfolding** *Interp\_def* **using** *less\_imp\_production\_subseteq\_interp*  
**by** (*metis le\_imp\_less\_or\_eq le\_supI1 sup\_ge2*)

**lemma** *less\_imp\_Interp\_subseteq\_interp*:  $C < D \implies \text{Interp } C \subseteq \text{interp } D$   
**by** (*simp add: Interp\_def less\_eq\_imp\_interp\_subseteq\_interp less\_imp\_production\_subseteq\_interp*)

**lemma** *less\_eq\_imp\_Interp\_subseteq\_Interp*:  $C \leq D \implies \text{Interp } C \subseteq \text{Interp } D$   
**using** *Interp\_def less\_eq\_imp\_interp\_subseteq\_Interp less\_eq\_imp\_production\_subseteq\_Interp* **by** *auto*

**lemma** *not\_Interp\_to\_interp\_imp\_less*:  $A \notin \text{Interp } C \implies A \in \text{interp } D \implies C < D$   
**using** *less\_eq\_imp\_interp\_subseteq\_Interp not\_less* **by** *blast*

**lemma** *not\_interp\_to\_interp\_imp\_less*:  $A \notin \text{interp } C \implies A \in \text{interp } D \implies C < D$   
**using** *less\_eq\_imp\_interp\_subseteq\_interp not\_less* **by** *blast*

**lemma** *not\_Interp\_to\_Interp\_imp\_less*:  $A \notin \text{Interp } C \implies A \in \text{Interp } D \implies C < D$   
**using** *less\_eq\_imp\_Interp\_subseteq\_Interp not\_less* **by** *blast*

**lemma** *not\_interp\_to\_Interp\_imp\_le*:  $A \notin \text{interp } C \implies A \in \text{Interp } D \implies C \leq D$   
**using** *less\_imp\_Interp\_subseteq\_interp not\_less* **by** *blast*

**definition** *INTERP* :: 'a *interp* **where**  
*INTERP* =  $(\bigcup C \in N. \text{production } C)$

**lemma** *interp\_subseteq\_INTERP*:  $\text{interp } C \subseteq \text{INTERP}$   
**unfolding** *interp\_def INTERP\_def* **by** (*auto simp: production\_unfold*)

**lemma** *production\_subseteq\_INTERP*:  $\text{production } C \subseteq \text{INTERP}$   
**unfolding** *INTERP\_def* **using** *production\_unfold* **by** *blast*

**lemma** *Interp\_subseteq\_INTERP*:  $\text{Interp } C \subseteq \text{INTERP}$   
**by** (*simp add: Interp\_def interp\_subseteq\_INTERP production\_subseteq\_INTERP*)

**lemma** *produces\_imp\_in\_interp*:  
**assumes** *a\_in\_c*:  $\text{Neg } A \in\# C$  **and** *d*: *produces*  $D A$   
**shows**  $A \in \text{interp } C$   
**by** (*metis Interp\_def Max\_pos\_neg\_less\_multiset UnCI a\_in\_c d not\_interp\_to\_Interp\_imp\_le not\_less producesD singletonI*)

**lemma** *neg\_notin\_Interp\_not\_produce*:  $\text{Neg } A \in\# C \implies A \notin \text{Interp } D \implies C \leq D \implies \neg \text{produces } D'' A$   
**using** *less\_eq\_imp\_interp\_subseteq\_Interp produces\_imp\_in\_interp* **by** *blast*

**lemma** *in\_production\_imp\_produces*:  $A \in \text{production } C \implies \text{produces } C A$   
**using** *productive\_imp\_produces\_Max\_atom* **by** *fastforce*

**lemma** *not\_produces\_imp\_notin\_production*:  $\neg \text{produces } C A \implies A \notin \text{production } C$   
**using** *in\_production\_imp\_produces* **by** *blast*

**lemma** *not\_produces\_imp\_notin\_interp*:  $(\bigwedge D. \neg \text{produces } D A) \implies A \notin \text{interp } C$   
**unfolding** *interp\_def* **by** (*fast intro!: in\_production\_imp\_produces*)

The results below corresponds to Lemma 3.4.

**lemma** *Interp\_imp\_general*:  
**assumes**  
*c\_le\_d*:  $C \leq D$  **and**  
*d\_lt\_d'*:  $D < D'$  **and**  
*c\_at\_d*:  $\text{Interp } D \models C$  **and**  
*subs*:  $\text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$



**shows**  $(\bigcup C \in CC. \text{production } C) \models C$   
**proof** (cases  $\exists A. \text{Pos } A \in \# C \wedge A \in \text{Interp } D$ )  
**case** *True*  
**then obtain**  $A$  **where**  $a\_in\_c: \text{Pos } A \in \# C$  **and**  $a\_at\_d: A \in \text{Interp } D$   
**by** *blast*  
**from**  $a\_at\_d$  **have**  $A \in \text{interp } D'$   
**using**  $d\_lt\_d'$  *less\_imp\_Interp\_subseteq\_interp* **by** *blast*  
**then show** *?thesis*  
**using**  $subs\ a\_in\_c$  **by** (*blast dest: contra\_subsetD*)  
**next**  
**case** *False*  
**then obtain**  $A$  **where**  $a\_in\_c: \text{Neg } A \in \# C$  **and**  $A \notin \text{Interp } D$   
**using**  $c\_at\_d$  *unfolding true\_cls\_def* **by** *blast*  
**then have**  $\bigwedge D''. \neg \text{produces } D'' A$   
**using**  $c\_le\_d$  *neg\_notin\_Interp\_not\_produce* **by** *simp*  
**then show** *?thesis*  
**using**  $a\_in\_c\ subs\ not\_produces\_imp\_notin\_production$  **by** *auto*  
**qed**

**lemma** *Interp\_imp\_interp*:  $C \leq D \implies D < D' \implies \text{Interp } D \models C \implies \text{interp } D' \models C$   
**using** *interp\_def Interp\_imp\_general* **by** *simp*

**lemma** *Interp\_imp\_Interp*:  $C \leq D \implies D \leq D' \implies \text{Interp } D \models C \implies \text{Interp } D' \models C$   
**using** *Interp\_as\_UNION interp\_subseteq\_Interp Interp\_imp\_general* **by** (*metis antisym\_conv2*)

**lemma** *Interp\_imp\_INTERP*:  $C \leq D \implies \text{Interp } D \models C \implies \text{INTERP} \models C$   
**using** *INTERP\_def interp\_subseteq\_INTERP Interp\_imp\_general* [*OF - le\_multiset\_right\_total*] **by** *simp*

**lemma** *interp\_imp\_general*:  
**assumes**  
 $c\_le\_d: C \leq D$  **and**  
 $d\_le\_d': D \leq D'$  **and**  
 $c\_at\_d: \text{interp } D \models C$  **and**  
 $subs: \text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$   
**shows**  $(\bigcup C \in CC. \text{production } C) \models C$   
**proof** (cases  $\exists A. \text{Pos } A \in \# C \wedge A \in \text{interp } D$ )  
**case** *True*  
**then obtain**  $A$  **where**  $a\_in\_c: \text{Pos } A \in \# C$  **and**  $a\_at\_d: A \in \text{interp } D$   
**by** *blast*  
**from**  $a\_at\_d$  **have**  $A \in \text{interp } D'$   
**using**  $d\_le\_d'$  *less\_eq\_imp\_interp\_subseteq\_interp* **by** *blast*  
**then show** *?thesis*  
**using**  $subs\ a\_in\_c$  **by** (*blast dest: contra\_subsetD*)  
**next**  
**case** *False*  
**then obtain**  $A$  **where**  $a\_in\_c: \text{Neg } A \in \# C$  **and**  $A \notin \text{interp } D$   
**using**  $c\_at\_d$  *unfolding true\_cls\_def* **by** *blast*  
**then have**  $\bigwedge D''. \neg \text{produces } D'' A$   
**using**  $c\_le\_d$  **by** (*auto dest: produces\_imp\_in\_interp less\_eq\_imp\_interp\_subseteq\_interp*)  
**then show** *?thesis*  
**using**  $a\_in\_c\ subs\ not\_produces\_imp\_notin\_production$  **by** *auto*  
**qed**

**lemma** *interp\_imp\_interp*:  $C \leq D \implies D \leq D' \implies \text{interp } D \models C \implies \text{interp } D' \models C$   
**using** *interp\_def interp\_imp\_general* **by** *simp*

**lemma** *interp\_imp\_Interp*:  $C \leq D \implies D \leq D' \implies \text{interp } D \models C \implies \text{Interp } D' \models C$   
**using** *Interp\_as\_UNION interp\_subseteq\_Interp* [*of D*] *interp\_imp\_general* **by** *simp*

**lemma** *interp\_imp\_INTERP*:  $C \leq D \implies \text{interp } D \models C \implies \text{INTERP} \models C$   
**using** *INTERP\_def interp\_subseteq\_INTERP interp\_imp\_general* *linear* **by** *metis*

**lemma** *productive\_imp\_not\_interp*:  $\text{productive } C \implies \neg \text{interp } C \models C$

**unfolding** *production\_unfold* **by** *simp*

This corresponds to Lemma 3.3:

**lemma** *productive\_imp\_Interp*:  
**assumes** *productive C*  
**shows** *Interp C*  $\models$  *C*  
**proof** –  
**obtain** *A* **where** *a: produces C A*  
**using** *assms productive\_imp\_produces\_Max\_atom* **by** *blast*  
**then have** *a.in.c: Pos A*  $\in$   $\#$  *C*  
**by** (*rule produces\_imp\_Pos.in.lits*)  
**moreover have** *A*  $\in$  *Interp C*  
**using** *a less\_eq\_imp\_production\_subseteq\_Interp* **by** *blast*  
**ultimately show** *?thesis*  
**by** *fast*  
**qed**

**lemma** *productive\_imp\_INTERP*: *productive C*  $\implies$  *INTERP*  $\models$  *C*  
**by** (*fast intro: productive\_imp\_Interp Interp\_imp\_INTERP*)

This corresponds to Lemma 3.5:

**lemma** *max\_pos\_imp\_Interp*:  
**assumes** *C*  $\in$  *N* **and** *C*  $\neq$   $\{\#\}$  **and** *Max.mset C = Pos A* **and** *S C = \{\#\}*  
**shows** *Interp C*  $\models$  *C*  
**proof** (*cases productive C*)  
**case** *True*  
**then show** *?thesis*  
**by** (*fast intro: productive\_imp\_Interp*)  
**next**  
**case** *False*  
**then have** *interp C*  $\models$  *C*  
**using** *assms unfolding production\_unfold* **by** *simp*  
**then show** *?thesis*  
**unfolding** *Interp\_def* **using** *False* **by** *auto*  
**qed**

The following results correspond to Lemma 3.6:

**lemma** *max\_atm\_imp\_Interp*:  
**assumes**  
*c.in.n: C*  $\in$  *N* **and**  
*pos.in: Pos A*  $\in$   $\#$  *C* **and**  
*max\_atm: A = Max (atms.of C)* **and**  
*s.c.e: S C = \{\#\}*  
**shows** *Interp C*  $\models$  *C*  
**proof** (*cases Neg A*  $\in$   $\#$  *C*)  
**case** *True*  
**then show** *?thesis*  
**using** *pos.in pos\_neg\_in\_imp\_true* **by** *metis*  
**next**  
**case** *False*  
**moreover have** *ne: C*  $\neq$   $\{\#\}$   
**using** *pos.in* **by** *auto*  
**ultimately have** *Max.mset C = Pos A*  
**using** *max\_atm* **using** *Max.in.lits Max.lit.eq\_pos\_or\_neg\_Max\_atm* **by** *metis*  
**then show** *?thesis*  
**using** *ne c.in\_n s.c.e* **by** (*blast intro: max\_pos\_imp\_Interp*)  
**qed**

**lemma** *not\_Interp\_imp\_general*:  
**assumes**  
*d'\_le.d: D'*  $\leq$  *D* **and**  
*in\_n.or\_max\_gt: D'  $\in$  N  $\wedge$  S D' = \{\#\}  $\vee$  Max (atms.of D') < Max (atms.of D)* **and**  
*d'\_at.d:  $\neg$  Interp D  $\models$  D'* **and**

```

d.lt.c:  $D < C$  and
subs:  $\text{interp } C \subseteq (\bigcup C \in CC. \text{ production } C)$ 
shows  $\neg (\bigcup C \in CC. \text{ production } C) \models D'$ 
proof -
{
  assume cc.blw.d':  $(\bigcup C \in CC. \text{ production } C) \models D'$ 
  have Interp D  $\subseteq (\bigcup C \in CC. \text{ production } C)$ 
    using less_imp_Interp_subseteq_interp d.lt.c subs by blast
  then obtain A where a.in.d':  $\text{Pos } A \in \# D'$  and a.blw.cc:  $A \in (\bigcup C \in CC. \text{ production } C)$ 
    using cc.blw.d' d'.at.d false_to_true_imp_ex_pos by metis
  from a.in.d' have a.at.d:  $A \notin \text{Interp } D$ 
    using d'.at.d by fast
  from a.blw.cc obtain C' where prod.c':  $\text{production } C' = \{A\}$ 
    by (fast intro!: in_production_imp_produces)
  have max.c':  $\text{Max } (\text{atms\_of } C') = A$ 
    using prod.c' productive_imp_produces_Max_atom by force
  have leq.dc':  $D \leq C'$ 
    using a.at.d d'.at.d prod.c' by (auto simp: Interp_def intro: not_interp_to_Interp_imp_le)
  then have D'  $\leq C'$ 
    using d'.le.d order_trans by blast
  then have max.d':  $\text{Max } (\text{atms\_of } D') = A$ 
    using a.in.d' max.c' by (fast intro: pos_lit_in_atms_of le_multiset_Max_in_imp_Max)

  {
    assume D'  $\in N \wedge S D' = \{\#\}$ 
    then have Interp D'  $\models D'$ 
      using a.in.d' max.d' by (blast intro: max_atm_imp_Interp)
    then have Interp D  $\models D'$ 
      using d'.le.d by (auto intro: Interp_imp_Interp simp: less_eq_multiset_def)
    then have False
      using d'.at.d by satx
  }
  moreover
  {
    assume Max (atms_of D')  $< \text{Max } (\text{atms\_of } D)$ 
    then have False
      using max.d' leq.dc' max.c' d'.le.d
      by (metis le_imp_less_or_eq le_multiset_empty_right less_eq_Max_atms_of less_imp_not_less)
  }
  ultimately have False
    using in_n_or_max_gt by satx
}
then show ?thesis
  by satx
qed

```

**lemma** not\_Interp\_imp\_not\_interp:

$$D' \leq D \implies D' \in N \wedge S D' = \{\#\} \vee \text{Max } (\text{atms\_of } D') < \text{Max } (\text{atms\_of } D) \implies \neg \text{Interp } D \models D' \implies D < C \implies \neg \text{interp } C \models D'$$

using interp\_def not\_Interp\_imp\_general by simp

**lemma** not\_Interp\_imp\_not\_INTERP:

$$D' \leq D \implies D' \in N \wedge S D' = \{\#\} \vee \text{Max } (\text{atms\_of } D') < \text{Max } (\text{atms\_of } D) \implies \neg \text{Interp } D \models D' \implies D < C \implies \neg \text{Interp } C \models D'$$

using Interp\_as\_UNION interp\_subseteq\_Interp not\_Interp\_imp\_general by metis

**lemma** not\_Interp\_imp\_not\_INTERP:

$$D' \leq D \implies D' \in N \wedge S D' = \{\#\} \vee \text{Max } (\text{atms\_of } D') < \text{Max } (\text{atms\_of } D) \implies \neg \text{Interp } D \models D' \implies \neg \text{INTERP } \models D'$$

using INTERP\_def interp\_subseteq\_INTERP not\_Interp\_imp\_general[OF - - le\_multiset\_right\_total] by simp

Lemma 3.7 is a problem child. It is stated below but not proved; instead, a counterexample is displayed. This is not much of a problem, because it is not invoked in the rest of the chapter.

```

lemma
  assumes  $D \in N$  and  $\bigwedge D'. D' < D \implies \text{Interp } D' \models C$ 
  shows  $\text{interp } D \models C$ 
oops

lemma
  assumes  $d: D = \{\#\}$  and  $n: N = \{D, C\}$  and  $c: C = \{\#Pos A\# \}$ 
  shows  $D \in N$  and  $\bigwedge D'. D' < D \implies \text{Interp } D' \models C$  and  $\neg \text{interp } D \models C$ 
  using  $n$  unfolding  $d$   $c$   $\text{interp\_def}$  by auto

end

end

end

```

## 10 Ground Unordered Resolution Calculus

```

theory Unordered_Ground_Resolution
  imports Inference_System Ground_Resolution_Model
begin

```

Unordered ground resolution is one of the two inference systems studied in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter.

### 10.1 Inference Rule

Unordered ground resolution consists of a single rule, called *unord\_resolve* below, which is sound and counterexample-reducing.

```

locale ground_resolution_without_selection
begin

sublocale ground_resolution_with_selection where  $S = \lambda_. \{\#\}$ 
  by unfold_locales auto

inductive unord_resolve :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool where
  unord_resolve ( $C + \text{replicate\_mset } (Suc\ n) (Pos\ A)$ ) ( $\text{add\_mset } (Neg\ A)\ D$ ) ( $C + D$ )

lemma unord_resolve_sound: unord_resolve  $C\ D\ E \implies I \models C \implies I \models D \implies I \models E$ 
  using unord_resolve.cases by fastforce

```

The following result corresponds to Theorem 3.8, except that the conclusion is strengthened slightly to make it fit better with the counterexample-reducing inference system framework.

```

theorem unord_resolve_counterex_reducing:
  assumes
     $ec\_ni\_n: \{\#\} \notin N$  and
     $c\_in\_n: C \in N$  and
     $c\_cex: \neg \text{INTERP } N \models C$  and
     $c\_min: \bigwedge D. D \in N \implies \neg \text{INTERP } N \models D \implies C \leq D$ 
  obtains  $D\ E$  where
     $D \in N$ 
     $\text{INTERP } N \models D$ 
    productive  $N\ D$ 
    unord_resolve  $D\ C\ E$ 
     $\neg \text{INTERP } N \models E$ 
     $E < C$ 
proof –
  have  $c\_ne: C \neq \{\#\}$ 
    using  $c\_in\_n$   $ec\_ni\_n$  by blast
  have  $\exists A. A \in \text{atms\_of } C \wedge A = \text{Max } (\text{atms\_of } C)$ 
    using  $c\_ne$  by (blast intro: Max\_in\_lits atm\_of\_Max\_lit atm\_of\_lit\_in\_atms\_of)

```

```

then have  $\exists A. \text{Neg } A \in \# C$ 
  using c_ne c.in.n c.cex c.min Max.in.lits Max.lit.eq_pos_or_neg_Max.atm max_pos_imp_Interp
  Interp_imp_INTERP by metis
then obtain  $A$  where  $\text{neg\_a\_in\_c}: \text{Neg } A \in \# C$ 
  by blast
then obtain  $C'$  where  $c: C = \text{add\_mset } (\text{Neg } A) C'$ 
  using insert_DiffM by metis
have  $A \in \text{INTERP } N$ 
  using neg_a_in_c c.cex[unfolded true_cls_def] by fast
then obtain  $D$  where  $d0: \text{produces } N D A$ 
  unfolding INTERP_def by (metis UN_E not_produces_imp_notin_production)
have  $\text{prod\_d}: \text{productive } N D$ 
  unfolding  $d0$  by simp
then have  $d.\text{in}.n: D \in N$ 
  using productive_in_N by fast
have  $d.\text{true}: \text{INTERP } N \models D$ 
  using prod_d productive_imp_INTERP by blast

obtain  $D' \text{ AAA}$  where
   $d: D = D' + \text{AAA}$  and
   $d': D' = \{\#L \in \# D. L \neq \text{Pos } A\}$  and
   $aa: \text{AAA} = \{\#L \in \# D. L = \text{Pos } A\}$ 
  using multiset_partition_union_commute by metis
have  $d'_{\text{subs}}: \text{set.mset } D' \subseteq \text{set.mset } D$ 
  unfolding  $d'$  by auto
have  $\neg \text{Neg } A \in \# D$ 
  using  $d0$  by (blast dest: produces_imp_neg_notin_lits)
then have  $\text{neg\_a\_ni\_d}': \neg \text{Neg } A \in \# D'$ 
  using  $d'_{\text{subs}}$  by auto
have  $a.\text{ni}.d': A \notin \text{atms\_of } D'$ 
  using  $d' \text{ neg\_a\_ni\_d}'$  by (auto dest: atm_imp_pos_or_neg_lit)
have  $\exists n. \text{AAA} = \text{replicate\_mset } (\text{Suc } n) (\text{Pos } A)$ 
  using  $aa d0 \text{ not0.implies\_Suc produces\_imp\_Pos\_in\_lits[of } N]$ 
  by (simp add: filter_eq_replicate_mset del: replicate_mset_Suc)
then have  $\text{res.e}: \text{unord\_resolve } D C (D' + C')$ 
  unfolding  $c d$  by (fastforce intro: unord_resolve.intros)

have  $d'_{\text{le}.d}: D' \leq D$ 
  unfolding  $d$  by simp
have  $a.\text{max}.d: A = \text{Max } (\text{atms\_of } D)$ 
  using  $d0 \text{ productive\_imp\_produces\_Max\_atom}$  by auto
then have  $D' \neq \{\#\} \implies \text{Max } (\text{atms\_of } D') \leq A$ 
  using  $d'_{\text{le}.d}$  by (blast intro: less_eq_Max_atms_of)
moreover have  $D' \neq \{\#\} \implies \text{Max } (\text{atms\_of } D') \neq A$ 
  using  $a.\text{ni}.d' \text{ Max.in}$  by (blast intro: atms_empty_iff_empty[THEN iffD1])
ultimately have  $\text{max}.d'_{\text{lt}.a}: D' \neq \{\#\} \implies \text{Max } (\text{atms\_of } D') < A$ 
  using dual_order.strict_iff_order by blast

have  $\neg \text{interp } N D \models D$ 
  using  $d0 \text{ productive\_imp\_not\_interp}$  by blast
then have  $\neg \text{Interp } N D \models D'$ 
  unfolding  $d0 d' \text{ Interp\_def true_cls\_def}$  by (auto simp: true_lit_def simp del: not_gr_zero)
then have  $\neg \text{INTERP } N \models D'$ 
  using  $a.\text{max}.d d'_{\text{le}.d} \text{ max}.d'_{\text{lt}.a} \text{ not\_Interp\_imp\_not\_INTERP}$  by blast
moreover have  $\neg \text{INTERP } N \models C'$ 
  using c_cex unfolding c by simp
ultimately have  $e.\text{cex}: \neg \text{INTERP } N \models D' + C'$ 
  by simp

have  $\bigwedge B. B \in \text{atms\_of } D' \implies B \leq A$ 
  using  $d0 d'_{\text{subs}} \text{ contra\_subsetD lits\_subsetq\_imp\_atms\_subsetq produces\_imp\_atms\_leq}$  by metis
then have  $\bigwedge L. L \in \# D' \implies L < \text{Neg } A$ 
  using  $\text{neg\_a\_ni}.d' \text{ antisym.conv1 atms\_less\_eq\_imp\_lit\_less\_eq\_neg}$  by metis

```

```

then have lt_cex:  $D' + C' < C$ 
  by (force intro: add.commute simp: c less_multisetDM intro: exI[of - {#Neg A#}])

from d_in_n d_true prod_d res_e e_cex lt_cex show ?thesis ..
qed

```

## 10.2 Inference System

Lemma 3.9 and Corollary 3.10 are subsumed in the counterexample-reducing inference system framework, which is instantiated below.

**definition** *unord\_Γ* :: 'a inference set **where**

```

unord_Γ = {Infer {#C#} D E | C D E. unord_resolve C D E}

```

**sublocale** *unord\_Γ\_sound\_counterex\_reducing?*:

```

sound_counterex_reducing_inference_system unord_Γ INTERP

```

**proof** *unfold\_locales*

```

fix D E and N :: ('b :: wellorder) clause set

```

```

assume {#} ∉ N and D ∈ N and ¬ INTERP N ⊨ D and ∧C. C ∈ N ⇒ ¬ INTERP N ⊨ C ⇒ D ≤ C

```

```

then obtain C E where

```

```

  c_in_n: C ∈ N and

```

```

  c_true: INTERP N ⊨ C and

```

```

  res_e: unord_resolve C D E and

```

```

  e_cex: ¬ INTERP N ⊨ E and

```

```

  e_lt_d: E < D

```

```

using unord_resolve_counterex_reducing by (metis (no_types))

```

```

from c_in_n have set_mset {#C#} ⊆ N

```

```

  by auto

```

```

moreover have Infer {#C#} D E ∈ unord_Γ

```

```

  unfolding unord_Γ_def using res_e by blast

```

```

ultimately show

```

```

  ∃ CC E. set_mset CC ⊆ N ∧ INTERP N ⊨m CC ∧ Infer CC D E ∈ unord_Γ ∧ ¬ INTERP N ⊨ E ∧ E < D

```

```

  using c_in_n c_true e_cex e_lt_d by blast

```

**next**

```

fix CC D E and I :: 'b interp

```

```

assume Infer CC D E ∈ unord_Γ and I ⊨m CC and I ⊨ D

```

```

then show I ⊨ E

```

```

  by (clarsimp simp: unord_Γ_def true_cls_mset_def) (erule unord_resolve_sound, auto)

```

**qed**

```

lemmas clausal_logic_compact = unord_Γ_sound_counterex_reducing.clausal_logic_compact

```

**end**

Theorem 3.12, compactness of clausal logic, has finally been derived for a concrete inference system:

```

lemmas clausal_logic_compact = ground_resolution_without_selection.clausal_logic_compact

```

**end**

## 11 Ground Ordered Resolution Calculus with Selection

**theory** *Ordered\_Ground\_Resolution*

```

imports Inference_System Ground_Resolution_Model

```

**begin**

Ordered ground resolution with selection is the second inference system studied in Section 3 (“Standard Resolution”) of Bachmair and Ganzinger’s chapter.

### 11.1 Inference Rule

Ordered ground resolution consists of a single rule, called *ord\_resolve* below. Like *unord\_resolve*, the rule is sound and counterexample-reducing. In addition, it is reductive.

**context** *ground\_resolution\_with\_selection*  
**begin**

The following inductive definition corresponds to Figure 2.

**definition** *maximal\_wrt* :: 'a  $\Rightarrow$  'a literal multiset  $\Rightarrow$  bool **where**  
*maximal\_wrt* A DA  $\equiv$  A = Max (atms\_of DA)

**definition** *strictly\_maximal\_wrt* :: 'a  $\Rightarrow$  'a literal multiset  $\Rightarrow$  bool **where**  
*strictly\_maximal\_wrt* A CA  $\longleftrightarrow (\forall B \in \text{atms\_of } CA. B < A)$

**inductive** *eligible* :: 'a list  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*eligible*: (S DA = negs (mset As))  $\vee$  (S DA = {#}  $\wedge$  length As = 1  $\wedge$  maximal\_wrt (As ! 0) DA)  $\implies$   
*eligible* As DA

**lemma** (S DA = negs (mset As)  $\vee$  S DA = {#}  $\wedge$  length As = 1  $\wedge$  maximal\_wrt (As ! 0) DA)  $\longleftrightarrow$   
*eligible* As DA  
**using** *eligible.intros* *ground\_resolution\_with\_selection.eligible.cases* *ground\_resolution\_with\_selection.axioms* **by** blast

**inductive**

*ord\_resolve* :: 'a clause list  $\Rightarrow$  'a clause  $\Rightarrow$  'a multiset list  $\Rightarrow$  'a list  $\Rightarrow$  'a clause  $\Rightarrow$  bool

**where**

*ord\_resolve*:  
length CAs = n  $\implies$   
length Cs = n  $\implies$   
length AAs = n  $\implies$   
length As = n  $\implies$   
n  $\neq$  0  $\implies$   
( $\forall i < n. \text{CAs} ! i = \text{Cs} ! i + \text{poss} (\text{AAs} ! i)$ )  $\implies$   
( $\forall i < n. \text{AAs} ! i \neq \{\#\}$ )  $\implies$   
( $\forall i < n. \forall A \in \# \text{AAs} ! i. A = \text{As} ! i$ )  $\implies$   
*eligible* As (D + negs (mset As))  $\implies$   
( $\forall i < n. \text{strictly\_maximal\_wrt} (\text{As} ! i) (\text{Cs} ! i)$ )  $\implies$   
( $\forall i < n. S (\text{CAs} ! i) = \{\#\}$ )  $\implies$   
*ord\_resolve* CAs (D + negs (mset As)) AAs As ( $\bigcup \# \text{mset Cs} + D$ )

**lemma** *ord\_resolve\_sound*:

**assumes**

*res\_e*: *ord\_resolve* CAs DA AAs As E **and**

*cc\_true*:  $I \models_m \text{mset CAs}$  **and**

*d\_true*:  $I \models DA$

**shows**  $I \models E$

**using** *res\_e*

**proof** (cases rule: *ord\_resolve.cases*)

**case** (*ord\_resolve* n Cs D)

**note** DA = *this*(1) **and** e = *this*(2) **and** cas\_len = *this*(3) **and** cs\_len = *this*(4) **and**  
as\_len = *this*(6) **and** cas = *this*(8) **and** aas\_ne = *this*(9) **and** a\_eq = *this*(10)

**show** ?thesis

**proof** (cases  $\forall A \in \text{set As}. A \in I$ )

**case** True

**then have**  $\neg I \models \text{negs} (\text{mset As})$

**unfolding** *true\_cls\_def* **by** fastforce

**then have**  $I \models D$

**using** *d\_true* DA **by** fast

**then show** ?thesis

**unfolding** e **by** blast

**next**

**case** False

**then obtain** i **where**

*a\_in\_aa*:  $i < n$  **and**

*a\_false*:  $\text{As} ! i \notin I$

**using** cas\_len as\_len **by** (metis in\_set\_conv\_nth)

```

have  $\neg I \models \text{poss } (AAs ! i)$ 
  using a_false a_eq aas_ne a_in_aa unfolding true_cls_def by auto
moreover have  $I \models CAs ! i$ 
  using a_in_aa cc_true unfolding true_cls_mset_def using cas_len by auto
ultimately have  $I \models Cs ! i$ 
  using cas a_in_aa by auto
then show ?thesis
  using a_in_aa cs_len unfolding e true_cls_def
  by (meson in_Union_mset_iff nth_mem_mset union_iff)
qed
qed

```

**lemma** *filter\_neg\_atm\_of\_S*:  $\{\# \text{Neg } (\text{atm\_of } L). L \in \# S C \# \} = S C$   
**by** (*simp add: Sselects\_neg lits*)

This corresponds to Lemma 3.13:

```

lemma ord_resolve_reductive:
  assumes ord_resolve CAs DA AAs As E
  shows  $E < DA$ 
  using assms
proof (cases rule: ord_resolve.cases)
  case (ord_resolve n Cs D)
  note  $DA = \text{this}(1)$  and  $e = \text{this}(2)$  and  $cas\_len = \text{this}(3)$  and  $cs\_len = \text{this}(4)$  and
     $ai\_len = \text{this}(6)$  and  $nz = \text{this}(7)$  and  $cas = \text{this}(8)$  and  $maxim = \text{this}(12)$ 

  show ?thesis
  proof (cases  $\bigcup \# mset Cs = \{\#\}$ )
    case True
    have  $\text{negs } (mset As) \neq \{\#\}$ 
      using nz ai_len by auto
    then show ?thesis
      unfolding True e DA by auto
  next
    case False

  define max_A_of-Cs where  $\text{max\_A\_of\_Cs} = \text{Max } (\text{atms\_of } (\bigcup \# mset Cs))$ 

  have
    mc_in:  $\text{max\_A\_of\_Cs} \in \text{atms\_of } (\bigcup \# mset Cs)$  and
    mc_max:  $\bigwedge B. B \in \text{atms\_of } (\bigcup \# mset Cs) \implies B \leq \text{max\_A\_of\_Cs}$ 
    using max_A_of-Cs_def False by auto

  then have  $\exists C\_max \in \text{set } Cs. \text{max\_A\_of\_Cs} \in \text{atms\_of } (C\_max)$ 
    by (metis atm_imp_pos_or_neg_lit in_Union_mset_iff neg_lit_in_atms_of pos_lit_in_atms_of
      set_mset_mset)
  then obtain max_i where
    cm_in_cas:  $\text{max\_i} < \text{length } CAs$  and
    mc_in_cm:  $\text{max\_A\_of\_Cs} \in \text{atms\_of } (Cs ! \text{max\_i})$ 
    using in_set_conv_nth[of _ CAs] by (metis cas_len cs_len in_set_conv_nth)

  define CA_max where  $CA\_max = CAs ! \text{max\_i}$ 
  define A_max where  $A\_max = As ! \text{max\_i}$ 
  define C_max where  $C\_max = Cs ! \text{max\_i}$ 

  have mc_lt_ma:  $\text{max\_A\_of\_Cs} < A\_max$ 
    using maxim cm_in_cas mc_in_cm cas_len unfolding strictly_maximal_wrt_def A_max_def by auto

  then have ucas_ne_neg_aa:  $(\bigcup \# mset Cs) \neq \text{negs } (mset As)$ 
    using mc_in mc_max mc_lt_ma cm_in_cas cas_len ai_len unfolding A_max_def
    by (metis atms_of_negs nth_mem set_mset_mset leD)
  moreover have ucas_lt_ma:  $\forall B \in \text{atms\_of } (\bigcup \# mset Cs). B < A\_max$ 
    using mc_max mc_lt_ma by fastforce
  moreover have  $\neg \text{Neg } A\_max \in \# (\bigcup \# mset Cs)$ 

```



```

    using ucas_lt_ma neg_lit_in_atms_of[of A_max  $\bigcup$  # mset Cs] by auto
  moreover have Neg A_max  $\in$  # negs (mset As)
    using cm_in_cas cas_len ai_len A_max_def by auto
  ultimately have ( $\bigcup$  # mset Cs) < negs (mset As)
    unfolding less_multiset_HO
    by (metis (no_types) atms_less_eq_imp_lit_less_eq_neg count_greater_zero_iff
        count_inI le_imp_less_or_eq less_imp_not_less not_le)
  then show ?thesis
    unfolding e DA by auto
qed
qed

```

This corresponds to Theorem 3.15:

**theorem** *ord\_resolve\_counterex\_reducing*:

```

assumes
  ec_ni_n: {#}  $\notin$  N and
  d_in_n: DA  $\in$  N and
  d_cex:  $\neg$  INTERP N  $\models$  DA and
  d_min:  $\bigwedge C. C \in N \implies \neg$  INTERP N  $\models$  C  $\implies$  DA  $\leq$  C
obtains CAs AAs As E where
  set CAs  $\subseteq$  N
  INTERP N  $\models_m$  mset CAs
   $\bigwedge CA. CA \in$  set CAs  $\implies$  productive N CA
  ord_resolve CAs DA AAs As E
   $\neg$  INTERP N  $\models$  E
  E < DA

```

**proof** –

```

  have d_ne: DA  $\neq$  {#}
    using d_in_n ec_ni_n by blast
  have  $\exists$  As. As  $\neq$  []  $\wedge$  negs (mset As)  $\leq$  # DA  $\wedge$  eligible As DA
proof (cases S DA = {#})
  assume s_d_e: S DA = {#}

```

```

define A where A = Max (atms_of DA)
define As where As = [A]
define D where D = DA – {#Neg A #}

```

```

have na_in_d: Neg A  $\in$  # DA
  unfolding A_def using s_d_e d_ne d_in_n d_cex d_min
  by (metis Max_in_lits Max_lit_eq_pos_or_neg Max_atm max_pos_imp_Interp Interp_imp_INTERP)
then have das: DA = D + negs (mset As) unfolding D_def As_def by auto
moreover from na_in_d have negs (mset As)  $\subseteq$  # DA
  by (simp add: As_def)
moreover have As ! 0 = Max (atms_of (D + negs (mset As)))
  using A_def As_def das by auto
then have eligible As DA
  using eligible s_d_e As_def das maximal_wrt_def by auto
ultimately show ?thesis
  using As_def by blast

```

**next**

```

  assume s_d_e: S DA  $\neq$  {#}

```

```

define As :: 'a list where
  As = list_of_mset {#atm_of L. L  $\in$  # S DA#}
define D :: 'a clause where
  D = DA – negs {#atm_of L. L  $\in$  # S DA#}

```

```

have As  $\neq$  [] unfolding As_def using s_d_e
  by (metis image_mset_is_empty_iff list_of_mset_empty)
moreover have da_sub_as: negs {#atm_of L. L  $\in$  # S DA#}  $\subseteq$  # DA
  using S_selects_subseteq by (auto simp: filter_neg_atm_of_S)
then have negs (mset As)  $\subseteq$  # DA
  unfolding As_def by auto

```

```

moreover have das:  $DA = D + \text{negs } (\text{mset } As)$ 
  using da_sub_as unfolding D_def As_def by auto
moreover have  $S \ DA = \text{negs } \{\# \text{atm\_of } L. L \in \# S \ DA\}$ 
  by (auto simp: filter_neg_atm_of_S)
then have  $S \ DA = \text{negs } (\text{mset } As)$ 
  unfolding As_def by auto
then have eligible As DA
  unfolding das using eligible by auto
ultimately show ?thesis
  by blast
qed
then obtain As :: 'a list where
  as_ne:  $As \neq []$  and
  negs_as_le_d:  $\text{negs } (\text{mset } As) \leq \# DA$  and
  s_d: eligible As DA
  by blast

define D :: 'a clause where
   $D = DA - \text{negs } (\text{mset } As)$ 

have set As  $\subseteq \text{INTERP } N$ 
  using d_cex negs_as_le_d by force
then have prod_ex:  $\forall A \in \text{set } As. \exists D. \text{produces } N \ D \ A$ 
  unfolding INTERP_def
  by (metis (no_types, lifting) INTERP_def subsetCE UN_E not_produces_imp_notin_production)
then have  $\bigwedge A. \exists D. \text{produces } N \ D \ A \longrightarrow A \in \text{set } As$ 
  using ec_ni_n by (auto intro: productive_in_N)
then have  $\bigwedge A. \exists D. \text{produces } N \ D \ A \longleftrightarrow A \in \text{set } As$ 
  using prod_ex by blast
then obtain CA_of where c_of0:  $\bigwedge A. \text{produces } N \ (CA\_of \ A) \ A \longleftrightarrow A \in \text{set } As$ 
  by metis
then have prod_c0:  $\forall A \in \text{set } As. \text{produces } N \ (CA\_of \ A) \ A$ 
  by blast

define C_of where
   $\bigwedge A. C\_of \ A = \{\# L \in \# CA\_of \ A. L \neq Pos \ A \#\}$ 
define Aj_of where
   $\bigwedge A. Aj\_of \ A = \text{image\_mset atm\_of } \{\# L \in \# CA\_of \ A. L = Pos \ A \#\}$ 

have pospos:  $\bigwedge LL \ A. \{\# Pos \ (\text{atm\_of } x). x \in \# \{\# L \in \# LL. L = Pos \ A \#\} \# \} = \{\# L \in \# LL. L = Pos \ A \#\}$ 
  by (metis (mono_tags, lifting) image_filter_cong literal.sel(1) multiset.map_ident)
have ca_of_c_of_aj_of:  $\bigwedge A. CA\_of \ A = C\_of \ A + \text{poss } (Aj\_of \ A)$ 
  using pospos[of - CA_of -] by (simp add: C_of_def Aj_of_def add commute multiset_partition)

define n :: nat where
   $n = \text{length } As$ 
define Cs :: 'a clause list where
   $Cs = \text{map } C\_of \ As$ 
define AAs :: 'a multiset list where
   $AAs = \text{map } Aj\_of \ As$ 
define CAs :: 'a literal multiset list where
   $CAs = \text{map } CA\_of \ As$ 

have m_nz:  $\bigwedge A. A \in \text{set } As \implies Aj\_of \ A \neq \{\#\}$ 
  unfolding Aj_of_def using prod_c0 produces_imp_Pos_in_lits
  by (metis (full_types) filter_mset_empty_conv image_mset_is_empty_iff)

have prod_c: productive N CA if ca_in:  $CA \in \text{set } CAs$  for CA
proof -
  obtain i where i_p:  $i < \text{length } CAs$   $CAs ! i = CA$ 
    using ca_in by (meson in_set_conv_nth)
  have production N (CA_of (As ! i)) = \{As ! i\}
    using i_p CAs_def prod_c0 by auto

```

```

    then show productive N CA
      using i_p CAs_def by auto
qed
then have cs_subs_n: set CAs  $\subseteq$  N
  using productive_in_N by auto
have cs_true: INTERP N  $\models_m$  mset CAs
  unfolding true_cls_mset_def using prod_c productive_imp_INTERP by auto

have  $\bigwedge A. A \in \text{set } As \implies \neg \text{Neg } A \in \# \text{ CA\_of } A$ 
  using prod_c0 produces_imp_neg_notin_lits by auto
then have a_ni_c':  $\bigwedge A. A \in \text{set } As \implies A \notin \text{atms\_of } (C\_of A)$ 
  unfolding C_of_def using atm_imp_pos_or_neg_lit by force
have c'_le_c:  $\bigwedge A. C\_of A \leq \text{CA\_of } A$ 
  unfolding C_of_def by (auto intro: subset_eq_imp_le_multiset)
have a_max_c:  $\bigwedge A. A \in \text{set } As \implies A = \text{Max } (\text{atms\_of } (\text{CA\_of } A))$ 
  using prod_c0 productive_imp_produces_Max_atom[of N] by auto
then have  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) \leq A$ 
  using c'_le_c by (metis less_eq_Max_atms_of)
moreover have  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) \neq A$ 
  using a_ni_c' Max_in by (metis (no_types) atms_empty_iff_empty finite_atms_of)
ultimately have max_c'_lt_a:  $\bigwedge A. A \in \text{set } As \implies C\_of A \neq \{\#\} \implies \text{Max } (\text{atms\_of } (C\_of A)) < A$ 
  by (metis order.strict_iff_order)

have le_cs_as: length CAs = length As
  unfolding CAs_def by simp

have length CAs = n
  by (simp add: le_cs_as n_def)
moreover have length Cs = n
  by (simp add: Cs_def n_def)
moreover have length AAs = n
  by (simp add: AAs_def n_def)
moreover have length As = n
  using n_def by auto
moreover have n  $\neq$  0
  by (simp add: as.ne n_def)
moreover have  $\forall i. i < \text{length } AAs \longrightarrow (\forall A \in \# AAs ! i. A = As ! i)$ 
  using AAs_def Aj_of_def by auto

have  $\bigwedge x B. \text{production } N (C\_of x) = \{x\} \implies B \in \# \text{ CA\_of } x \implies B \neq \text{Pos } x \implies \text{atm\_of } B < x$ 
  by (metis atm_of_lit_in_atms_of insert_not_empty le_imp_less_or_eq Pos_atm_of_iff
    Neg_atm_of_iff pos_neg_in_imp_true produces_imp_Pos_in_lits produces_imp_atms_leq
    productive_imp_not_interp)
then have  $\bigwedge B A. A \in \text{set } As \implies B \in \# \text{ CA\_of } A \implies B \neq \text{Pos } A \implies \text{atm\_of } B < A$ 
  using prod_c0 by auto
have  $\forall i. i < \text{length } AAs \longrightarrow AAs ! i \neq \{\#\}$ 
  unfolding AAs_def using m_nz by simp
have  $\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)$ 
  unfolding CAs_def Cs_def AAs_def using ca_of_c_of_aj_of by (simp add: n_def)

moreover have  $\forall i < n. AAs ! i \neq \{\#\}$ 
  using  $\langle \forall i < \text{length } AAs. AAs ! i \neq \{\#\} \rangle$  calculation(3) by blast
moreover have  $\forall i < n. \forall A \in \# AAs ! i. A = As ! i$ 
  by (simp add:  $\langle \forall i < \text{length } AAs. \forall A \in \# AAs ! i. A = As ! i \rangle$  calculation(3))
moreover have eligible As DA
  using s_d by auto
then have eligible As (D + negs (mset As))
  using D_def negs_as_le_d by auto
moreover have  $\bigwedge i. i < \text{length } AAs \implies \text{strictly\_maximal\_wrt } (As ! i) ((Cs ! i))$ 
  by (simp add: C_of_def Cs_def  $\langle \bigwedge x B. \llbracket \text{production } N (C\_of x) = \{x\}; B \in \# \text{ CA\_of } x; B \neq \text{Pos } x \rrbracket \implies \text{atm\_of } B < x \rangle$ 
    atms_of_def calculation(3) n_def prod_c0 strictly_maximal_wrt_def)

have  $\forall i < n. \text{strictly\_maximal\_wrt } (As ! i) (Cs ! i)$ 

```

```

  by (simp add: ⟨ $\bigwedge i. i < \text{length } AAs \implies \text{strictly\_maximal\_wrt } (As ! i) (Cs ! i) \rangle \text{ calculation}(3)$ )
moreover have  $\forall CA \in \text{set } CAs. S \ CA = \{\#\}$ 
  using prod_c producesD productive_imp_produces_Max_literal by blast
have  $\forall CA \in \text{set } CAs. S \ CA = \{\#\}$ 
  using ⟨ $\forall CA \in \text{set } CAs. S \ CA = \{\#\}$ ⟩ by simp
then have  $\forall i < n. S \ (CAs ! i) = \{\#\}$ 
  using ⟨ $\text{length } CAs = n$ ⟩ nth_mem by blast
ultimately have res_e: ord_resolve CAs (D + negs (mset As)) AAs As ( $\bigcup \# \text{ mset } Cs + D$ )
  using ord_resolve by auto

have  $\bigwedge A. A \in \text{set } As \implies \neg \text{interp } N \ (CA\_of \ A) \models CA\_of \ A$ 
  by (simp add: prod_c0 producesD)
then have  $\bigwedge A. A \in \text{set } As \implies \neg \text{Interp } N \ (CA\_of \ A) \models C\_of \ A$ 
  unfolding prod_c0 C_of_def Interp_def true_cls_def using true_lit_def not_gr_zero prod_c0
  by auto
then have c'_at_n:  $\bigwedge A. A \in \text{set } As \implies \neg \text{INTERP } N \models C\_of \ A$ 
  using a_max_c c'_le_c max_c'_lt_a not_Interp_imp_not_INTERP unfolding true_cls_def
  by (metis true_cls_def true_cls_empty)

have  $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs$ 
  unfolding Cs_def true_cls_def by (auto dest!: c'_at_n)
moreover have  $\neg \text{INTERP } N \models D$ 
  using d_cex by (metis D_def add_diff_cancel_right' negs_as_le_d subset_mset.add_diff_assoc2
    true_cls_def union_iff)
ultimately have e_cex:  $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs + D$ 
  by simp

have  $\text{set } CAs \subseteq N$ 
  by (simp add: cs_subs_n)
moreover have  $\text{INTERP } N \models_m \text{ mset } CAs$ 
  by (simp add: cs_true)
moreover have  $\bigwedge CA. CA \in \text{set } CAs \implies \text{productive } N \ CA$ 
  by (simp add: prod_c)
moreover have ord_resolve CAs DA AAs As ( $\bigcup \# \text{ mset } Cs + D$ )
  using D_def negs_as_le_d res_e by auto
moreover have  $\neg \text{INTERP } N \models \bigcup \# \text{ mset } Cs + D$ 
  using e_cex by simp
moreover have  $(\bigcup \# \text{ mset } Cs + D) < DA$ 
  using calculation(4) ord_resolve_reductive by auto
ultimately show thesis
..
qed

lemma ord_resolve_atms_of_concl_subset:
  assumes ord_resolve CAs DA AAs As E
  shows  $\text{atms\_of } E \subseteq (\bigcup C \in \text{set } CAs. \text{atms\_of } C) \cup \text{atms\_of } DA$ 
  using assms
proof (cases rule: ord_resolve.cases)
case (ord_resolve n Cs D)
  note DA = this(1) and e = this(2) and cas_len = this(3) and cs_len = this(4) and cas = this(8)

  have  $\forall i < n. \text{set\_mset } (Cs ! i) \subseteq \text{set\_mset } (CAs ! i)$ 
    using cas by auto
  then have  $\forall i < n. Cs ! i \subseteq \# \bigcup \# \text{ mset } CAs$ 
    by (metis cas cas_len mset_subset_eq_add_left nth_mem_mset sum_mset.remove union_assoc)
  then have  $\forall C \in \text{set } Cs. C \subseteq \# \bigcup \# \text{ mset } CAs$ 
    using cs_len in_set_conv_nth[of _ Cs] by auto
  then have  $\text{set\_mset } (\bigcup \# \text{ mset } Cs) \subseteq \text{set\_mset } (\bigcup \# \text{ mset } CAs)$ 
    by auto (meson in_mset_sum_list2 mset_subset_eqD)
  then have  $\text{atms\_of } (\bigcup \# \text{ mset } Cs) \subseteq \text{atms\_of } (\bigcup \# \text{ mset } CAs)$ 
    by (meson lits_subseteq_imp_atms_subseteq mset_subset_eqD subsetI)
  moreover have  $\text{atms\_of } (\bigcup \# \text{ mset } CAs) = (\bigcup CA \in \text{set } CAs. \text{atms\_of } CA)$ 
    by (intro set_eqI iffI, simp_all,

```

```

metis in_mset_sum_list2 atm_imp_pos_or_neg_lit neg_lit_in_atms_of pos_lit_in_atms_of,
metis in_mset_sum_list atm_imp_pos_or_neg_lit neg_lit_in_atms_of pos_lit_in_atms_of)
ultimately have atms_of ( $\bigcup \# \text{ mset } Cs$ )  $\subseteq$  ( $\bigcup CA \in \text{ set } CAs. \text{ atms\_of } CA$ )
by auto
moreover have atms_of  $D \subseteq$  atms_of  $DA$ 
using  $DA$  by auto
ultimately show ?thesis
unfolding  $e$  by auto
qed

```

## 11.2 Inference System

Theorem 3.16 is subsumed in the counterexample-reducing inference system framework, which is instantiated below. Unlike its unordered cousin, ordered resolution is additionally a reductive inference system.

**definition**  $\text{ord}\Gamma :: 'a \text{ inference set}$  **where**

$\text{ord}\Gamma = \{\text{Infer } (\text{mset } CAs) \text{ } DA \text{ } E \mid CAs \text{ } DA \text{ } AAs \text{ } As \text{ } E. \text{ ord\_resolve } CAs \text{ } DA \text{ } AAs \text{ } As \text{ } E\}$

**sublocale**  $\text{ord}\Gamma \text{ sound\_counterex\_reducing?}$ :

$\text{sound\_counterex\_reducing\_inference\_system } \text{ground\_resolution\_with\_selection.ord}\Gamma \text{ } S$   
 $\text{ground\_resolution\_with\_selection.INTERP } S +$   
 $\text{reductive\_inference\_system } \text{ground\_resolution\_with\_selection.ord}\Gamma \text{ } S$

**proof**  $\text{unfold.locales}$

**fix**  $DA :: 'a \text{ clause}$  **and**  $N :: 'a \text{ clause set}$

**assume**  $\{\#\} \notin N$  **and**  $DA \in N$  **and**  $\neg \text{INTERP } N \models DA$  **and**  $\bigwedge C. C \in N \implies \neg \text{INTERP } N \models C \implies DA \leq C$

**then obtain**  $CAs \text{ } AAs \text{ } As \text{ } E$  **where**

$\text{dd\_sset\_n}: \text{ set } CAs \subseteq N$  **and**

$\text{dd\_true}: \text{INTERP } N \models_m \text{mset } CAs$  **and**

$\text{res\_e}: \text{ord\_resolve } CAs \text{ } DA \text{ } AAs \text{ } As \text{ } E$  **and**

$\text{e\_ce}: \neg \text{INTERP } N \models E$  **and**

$\text{e\_lt\_c}: E < DA$

**using**  $\text{ord\_resolve\_counterex\_reducing}[of \text{ } N \text{ } DA \text{ } thesis]$  **by**  $\text{auto}$

**have**  $\text{Infer } (\text{mset } CAs) \text{ } DA \text{ } E \in \text{ord}\Gamma$

**using**  $\text{res\_e}$  **unfolding**  $\text{ord}\Gamma\_def$  **by**  $(metis \text{ (mono\_tags, lifting) mem\_Collect\_eq})$

**then show**  $\exists CC \text{ } E. \text{ set\_mset } CC \subseteq N \wedge \text{INTERP } N \models_m CC \wedge \text{Infer } CC \text{ } DA \text{ } E \in \text{ord}\Gamma$   
 $\wedge \neg \text{INTERP } N \models E \wedge E < DA$

**using**  $\text{dd\_sset\_n } \text{dd\_true } \text{e\_ce} \text{ } \text{e\_lt\_c}$  **by**  $(metis \text{ set\_mset\_mset})$

**qed**  $(\text{auto simp: ord}\Gamma\_def \text{ intro: ord\_resolve\_sound ord\_resolve\_reductive})$

**lemmas**  $\text{clausal\_logic\_compact} = \text{ord}\Gamma \text{ sound\_counterex\_reducing.clausal\_logic\_compact}$

**end**

A second proof of Theorem 3.12, compactness of clausal logic:

**lemmas**  $\text{clausal\_logic\_compact} = \text{ground\_resolution\_with\_selection.clausal\_logic\_compact}$

**end**

## 12 Theorem Proving Processes

**theory**  $\text{Proving\_Process}$

**imports**  $\text{Unordered\_Ground\_Resolution Lazy\_List\_Chain}$

**begin**

This material corresponds to Section 4.1 (“Theorem Proving Processes”) of Bachmair and Ganzinger’s chapter.

The locale assumptions below capture conditions R1 to R3 of Definition 4.1.  $Rf$  denotes  $\mathcal{R}_{\mathcal{F}}$ ;  $Ri$  denotes  $\mathcal{R}_{\mathcal{I}}$ .

**locale**  $\text{redundancy\_criterion} = \text{inference\_system} +$

**fixes**

```

  Rf :: 'a clause set  $\Rightarrow$  'a clause set and
  Ri :: 'a clause set  $\Rightarrow$  'a inference set
assumes
  Ri_subset_Γ: Ri N  $\subseteq$  Γ and
  Rf_mono: N  $\subseteq$  N'  $\Longrightarrow$  Rf N  $\subseteq$  Rf N' and
  Ri_mono: N  $\subseteq$  N'  $\Longrightarrow$  Ri N  $\subseteq$  Ri N' and
  Rf_indep: N'  $\subseteq$  Rf N  $\Longrightarrow$  Rf N  $\subseteq$  Rf (N - N') and
  Ri_indep: N'  $\subseteq$  Rf N  $\Longrightarrow$  Ri N  $\subseteq$  Ri (N - N') and
  Rf_sat: satisfiable (N - Rf N)  $\Longrightarrow$  satisfiable N
begin

definition saturated_upto :: 'a clause set  $\Rightarrow$  bool where
  saturated_upto N  $\longleftrightarrow$  inferences_from (N - Rf N)  $\subseteq$  Ri N

inductive derive :: 'a clause set  $\Rightarrow$  'a clause set  $\Rightarrow$  bool (infix  $\triangleright$  50) where
  deduction_deletion: N - M  $\subseteq$  concls_of (inferences_from M)  $\Longrightarrow$  M - N  $\subseteq$  Rf N  $\Longrightarrow$  M  $\triangleright$  N

lemma derive_subset: M  $\triangleright$  N  $\Longrightarrow$  N  $\subseteq$  M  $\cup$  concls_of (inferences_from M)
  by (meson Diff_subset_conv derive.cases)

end

locale sat_preserving_redundancy_criterion =
  sat_preserving_inference_system Γ :: ('a :: wellorder) inference set + redundancy_criterion
begin

lemma deriv_sat_preserving:
  assumes
    deriv: chain (op  $\triangleright$ ) Ns and
    sat_n0: satisfiable (lhd Ns)
  shows satisfiable (Sup_llist Ns)
proof -
  have ns0: lnth Ns 0 = lhd Ns
    using deriv by (metis chain_not_lnull lhd_conv_lnth)
  have len_ns: llength Ns > 0
    using deriv by (case_tac Ns) simp+
  {
    fix DD
    assume fin: finite DD and sset_lun: DD  $\subseteq$  Sup_llist Ns
    then obtain k where dd_sset: DD  $\subseteq$  Sup_upto_llist Ns k
      using finite_Sup_llist_imp_Sup_upto_llist by blast
    have satisfiable (Sup_upto_llist Ns k)
    proof (induct k)
      case 0
      then show ?case
        using len_ns ns0 sat_n0 unfolding Sup_upto_llist_def true_cls_def by auto
    next
      case (Suc k)
      show ?case
    proof (cases enat (Suc k)  $\geq$  llength Ns)
      case True
      then have Sup_upto_llist Ns k = Sup_upto_llist Ns (Suc k)
        unfolding Sup_upto_llist_def using le_Suc_eq not_less by blast
      then show ?thesis
        using Suc by simp
    next
      case False
      then have lnth Ns k  $\triangleright$  lnth Ns (Suc k)
        using deriv by (auto simp: chain_lnth_rel)
      then have lnth Ns (Suc k)  $\subseteq$  lnth Ns k  $\cup$  concls_of (inferences_from (lnth Ns k))
        by (rule derive_subset)
      moreover have lnth Ns k  $\subseteq$  Sup_upto_llist Ns k
        unfolding Sup_upto_llist_def using False Suc_ile_eq linear by blast
    qed
  }

```

```

ultimately have lnth Ns (Suc k)
  ⊆ Sup_upto_llist Ns k ∪ concls_of (inferences_from (Sup_upto_llist Ns k))
  by clarsimp (metis UnCI UnE image_Un inferences_from_mono le_iff_sup)
moreover have Sup_upto_llist Ns (Suc k) = Sup_upto_llist Ns k ∪ lnth Ns (Suc k)
  unfolding Sup_upto_llist_def using False by (force elim: le_SucE)
moreover have
  satisfiable (Sup_upto_llist Ns k ∪ concls_of (inferences_from (Sup_upto_llist Ns k)))
  using Suc Γ_sat_preserving unfolding sat_preserving_inference_system_def by simp
ultimately show ?thesis
  by (metis le_iff_sup true_cls_union)
qed
qed
then have satisfiable DD
  using dd_sset unfolding Sup_upto_llist_def by (blast intro: true_cls_mono)
}
then show ?thesis
  using ground_resolution_without_selection.clausal_logic_compact[THEN iffD1] by metis
qed

```

This corresponds to Lemma 4.2:

**lemma**

**assumes** *deriv*:  $\text{chain } (op \triangleright) Ns$

**shows**

$Rf\_Sup\_subset\_Rf\_Liminf$ :  $Rf (Sup\_llist Ns) \subseteq Rf (Liminf\_llist Ns)$  **and**  
 $Ri\_Sup\_subset\_Ri\_Liminf$ :  $Ri (Sup\_llist Ns) \subseteq Ri (Liminf\_llist Ns)$  **and**  
 $sat\_deriv\_Liminf\_iff$ :  $\text{satisfiable } (Liminf\_llist Ns) \longleftrightarrow \text{satisfiable } (lhd Ns)$

**proof** –

```

{
  fix C i j
  assume
    c_in:  $C \in lnth Ns i$  and
    c_ni:  $C \notin Rf (Sup\_llist Ns)$  and
    j:  $j \geq i$  and
    j':  $enat j < llength Ns$ 
  from c_ni have c_ni':  $\bigwedge i. enat i < llength Ns \implies C \notin Rf (lnth Ns i)$ 
    using Rf_mono lnth_subset_Sup_llist Sup_llist_def by (blast dest: contra_subsetD)
  have C ∈ lnth Ns j
  using j j'
  proof (induct j)
    case 0
    then show ?case
      using c_in by blast
  next
    case (Suc k)
    then show ?case
    proof (cases  $i < Suc k$ )
      case True
      have  $i \leq k$ 
      using True by linarith
      moreover have  $enat k < llength Ns$ 
      using Suc.prem(2) Suc_ile_eq by (blast intro: dual_order.strict.implies_order)
      ultimately have c_in_k:  $C \in lnth Ns k$ 
      using Suc.hyps by blast
      have rel:  $lnth Ns k \triangleright lnth Ns (Suc k)$ 
      using Suc.prem deriv by (auto simp: chain_lnth_rel)
      then show ?thesis
      using c_in_k c_ni' Suc.prem(2) by cases auto
    next
      case False
      then show ?thesis
      using Suc c_in by auto
    qed
  qed
}

```

```

}
then have lu_ll: Sup_llist Ns - Rf (Sup_llist Ns)  $\subseteq$  Liminf_llist Ns
  unfolding Sup_llist_def Liminf_llist_def by blast
have rf: Rf (Sup_llist Ns - Rf (Sup_llist Ns))  $\subseteq$  Rf (Liminf_llist Ns)
  using lu_ll Rf_mono by simp
have ri: Ri (Sup_llist Ns - Rf (Sup_llist Ns))  $\subseteq$  Ri (Liminf_llist Ns)
  using lu_ll Ri_mono by simp
show Rf (Sup_llist Ns)  $\subseteq$  Rf (Liminf_llist Ns)
  using rf Rf_indep by blast
show Ri (Sup_llist Ns)  $\subseteq$  Ri (Liminf_llist Ns)
  using ri Ri_indep by blast

show satisfiable (Liminf_llist Ns)  $\longleftrightarrow$  satisfiable (lhd Ns)
proof
  assume satisfiable (lhd Ns)
  then have satisfiable (Sup_llist Ns)
    using deriv deriv_sat_preserving by simp
  then show satisfiable (Liminf_llist Ns)
    using true_cls_mono[OF Liminf_llist_subset_Sup_llist] by blast
next
  assume satisfiable (Liminf_llist Ns)
  then have satisfiable (Sup_llist Ns - Rf (Sup_llist Ns))
    using true_cls_mono[OF lu_ll] by blast
  then have satisfiable (Sup_llist Ns)
    using Rf_sat by blast
  then show satisfiable (lhd Ns)
    using deriv true_cls_mono lhd_subset_Sup_llist chain_not_lnull by metis
qed
qed

lemma
  assumes chain (op  $\triangleright$ ) Ns
  shows
    Rf_Liminf_eq_Rf_Sup: Rf (Liminf_llist Ns) = Rf (Sup_llist Ns) and
    Ri_Liminf_eq_Ri_Sup: Ri (Liminf_llist Ns) = Ri (Sup_llist Ns)
  using assms
  by (auto simp: Rf_Sup_subset_Rf_Liminf Rf_mono Ri_Sup_subset_Ri_Liminf Ri_mono
    Liminf_llist_subset_Sup_llist subset_antisym)

```

end

The assumption below corresponds to condition R4 of Definition 4.1.

```

locale effective_redundancy_criterion = redundancy_criterion +
  assumes Ri_effective:  $\gamma \in \Gamma \implies \text{concl\_of } \gamma \in N \cup \text{Rf } N \implies \gamma \in \text{Ri } N$ 
begin

```

```

definition fair_cls_seq :: 'a clause set llist  $\Rightarrow$  bool where
  fair_cls_seq Ns  $\longleftrightarrow$  (let N' = Liminf_llist Ns - Rf (Liminf_llist Ns) in
    concls_of (inferences_from N' - Ri N')  $\subseteq$  Sup_llist Ns  $\cup$  Rf (Sup_llist Ns))

```

end

```

locale sat_preserving_effective_redundancy_criterion =
  sat_preserving_inference_system  $\Gamma$  :: ('a :: wellorder) inference set +
  effective_redundancy_criterion
begin

```

```

sublocale sat_preserving_redundancy_criterion
..

```

The result below corresponds to Theorem 4.3.

```

theorem fair_derive_saturated_upto:
  assumes

```



```

    deriv: chain (op  $\triangleright$ ) Ns and
    fair: fair_clss_seq Ns
shows saturated_upto (Liminf_llist Ns)
unfolding saturated_upto_def
proof
  fix  $\gamma$ 
  let  $?N' = \text{Liminf\_llist } Ns - Rf (\text{Liminf\_llist } Ns)$ 
  assume  $\gamma: \gamma \in \text{inferences\_from } ?N'$ 
  show  $\gamma \in Ri (\text{Liminf\_llist } Ns)$ 
  proof (cases  $\gamma \in Ri ?N'$ )
    case True
    then show ?thesis
      using Ri_mono by blast
  next
    case False
    have concl_of (inferences_from  $?N' - Ri ?N'$ )  $\subseteq$  Sup_llist Ns  $\cup$  Rf (Sup_llist Ns)
      using fair unfolding fair_clss_seq_def Let_def .
    then have concl_of  $\gamma \in \text{Sup\_llist } Ns \cup Rf (\text{Sup\_llist } Ns)$ 
      using False  $\gamma$  by auto
    moreover
    {
      assume concl_of  $\gamma \in \text{Sup\_llist } Ns$ 
      then have  $\gamma \in Ri (\text{Sup\_llist } Ns)$ 
        using  $\gamma$  Ri_effective_inferences_from_def by blast
      then have  $\gamma \in Ri (\text{Liminf\_llist } Ns)$ 
        using deriv Ri_Sup_subset_Ri_Liminf by fast
    }
    moreover
    {
      assume concl_of  $\gamma \in Rf (\text{Sup\_llist } Ns)$ 
      then have concl_of  $\gamma \in Rf (\text{Liminf\_llist } Ns)$ 
        using deriv Rf_Sup_subset_Rf_Liminf by blast
      then have  $\gamma \in Ri (\text{Liminf\_llist } Ns)$ 
        using  $\gamma$  Ri_effective_inferences_from_def by auto
    }
    ultimately show  $\gamma \in Ri (\text{Liminf\_llist } Ns)$ 
      by blast
  qed
qed
end

```

This corresponds to the trivial redundancy criterion defined on page 36 of Section 4.1.

```

locale trivial_redundancy_criterion = inference_system
begin

```

```

definition Rf :: 'a clause set  $\Rightarrow$  'a clause set where
  Rf _ = {}

```

```

definition Ri :: 'a clause set  $\Rightarrow$  'a inference set where
  Ri N =  $\{\gamma. \gamma \in \Gamma \wedge \text{concl\_of } \gamma \in N\}$ 

```

```

sublocale effective_redundancy_criterion  $\Gamma$  Rf Ri
  by unfold_locales (auto simp: Rf_def Ri_def)

```

```

lemma saturated_upto_iff: saturated_upto N  $\longleftrightarrow$  concl_of (inferences_from N)  $\subseteq$  N
  unfolding saturated_upto_def inferences_from_def Rf_def Ri_def by auto

```

```

end

```

The following lemmas corresponds to the standard extension of a redundancy criterion defined on page 38 of Section 4.1.

```

lemma redundancy_criterion_standard_extension:

```

```

assumes  $\Gamma \subseteq \Gamma'$  and redundancy_criterion  $\Gamma$  Rf Ri
shows redundancy_criterion  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ 
using assms unfolding redundancy_criterion_def by (intro conjI) ((auto simp: rev_subsetD)[5], sat)

lemma redundancy_criterion_standard_extension_saturated_upto_iff:
assumes  $\Gamma \subseteq \Gamma'$  and redundancy_criterion  $\Gamma$  Rf Ri
shows redundancy_criterion.saturated_upto  $\Gamma$  Rf Ri M  $\longleftrightarrow$ 
  redundancy_criterion.saturated_upto  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ M$ 
using assms redundancy_criterion.saturated_upto_def redundancy_criterion.saturated_upto_def
  redundancy_criterion_standard_extension
unfolding inference_system.inferences_from_def by blast

lemma redundancy_criterion_standard_extension_effective:
assumes  $\Gamma \subseteq \Gamma'$  and effective_redundancy_criterion  $\Gamma$  Rf Ri
shows effective_redundancy_criterion  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$ 
using assms redundancy_criterion_standard_extension[of  $\Gamma$ ]
unfolding effective_redundancy_criterion_def effective_redundancy_criterion_axioms_def by auto

lemma redundancy_criterion_standard_extension_fair_iff:
assumes  $\Gamma \subseteq \Gamma'$  and effective_redundancy_criterion  $\Gamma$  Rf Ri
shows effective_redundancy_criterion.fair_clsseq  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns \longleftrightarrow$ 
  effective_redundancy_criterion.fair_clsseq  $\Gamma$  Rf Ri Ns
using assms redundancy_criterion_standard_extension_effective[of  $\Gamma$   $\Gamma'$  Rf Ri]
  effective_redundancy_criterion.fair_clsseq_def[of  $\Gamma$  Rf Ri Ns]
  effective_redundancy_criterion.fair_clsseq_def[of  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns$ ]
unfolding inference_system.inferences_from_def Let_def by auto

theorem redundancy_criterion_standard_extension_fair_derive_saturated_upto:
assumes
  subs:  $\Gamma \subseteq \Gamma'$  and
  red: redundancy_criterion  $\Gamma$  Rf Ri and
  red': sat_preserving_effective_redundancy_criterion  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$  and
  deriv: chain (redundancy_criterion.derive  $\Gamma'$  Rf) Ns and
  fair: effective_redundancy_criterion.fair_clsseq  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))\ Ns$ 
shows redundancy_criterion.saturated_upto  $\Gamma$  Rf Ri (Liminf_llist Ns)
proof –
  have redundancy_criterion.saturated_upto  $\Gamma'$  Rf  $(\lambda N. Ri\ N \cup (\Gamma' - \Gamma))$  (Liminf_llist Ns)
    by (rule sat_preserving_effective_redundancy_criterion.fair_derive_saturated_upto
      [OF red' deriv fair])
  then show ?thesis
    by (rule redundancy_criterion_standard_extension_saturated_upto_iff[THEN iffD2, OF subs red])
qed

end

```

## 13 The Standard Redundancy Criterion

```

theory Standard_Redundancy
imports Proving_Process
begin

```

This material is based on Section 4.2.2 (“The Standard Redundancy Criterion”) of Bachmair and Ganzinger’s chapter.

```

locale standard_redundancy_criterion =
  inference_system  $\Gamma$  for  $\Gamma :: ('a :: wellorder)\ inference\ set$ 
begin

```

```

abbreviation redundant_infer :: 'a clause set  $\Rightarrow$  'a inference  $\Rightarrow$  bool where
  redundant_infer  $N\ \gamma \equiv$ 
   $\exists DD. set\_mset\ DD \subseteq N \wedge (\forall I. I \models_m DD + side\_prems\_of\ \gamma \longrightarrow I \models concl\_of\ \gamma)$ 
   $\wedge (\forall D. D \in \# DD \longrightarrow D < main\_prem\_of\ \gamma)$ 

```

**definition**  $Rf :: 'a \text{ clause set} \Rightarrow 'a \text{ clause set}$  **where**

$Rf\ N = \{C. \exists DD. \text{set\_mset } DD \subseteq N \wedge (\forall I. I \models_m DD \longrightarrow I \models C) \wedge (\forall D. D \in\# DD \longrightarrow D < C)\}$

**definition**  $Ri :: 'a \text{ clause set} \Rightarrow 'a \text{ inference set}$  **where**

$Ri\ N = \{\gamma \in \Gamma. \text{redundant\_infer } N\ \gamma\}$

**lemma** *tautology\_redundant*:

**assumes**  $Pos\ A \in\# C$

**assumes**  $Neg\ A \in\# C$

**shows**  $C \in Rf\ N$

**proof** –

**have**  $\text{set\_mset } \{\#\} \subseteq N \wedge (\forall I. I \models_m \{\#\} \longrightarrow I \models C) \wedge (\forall D. D \in\# \{\#\} \longrightarrow D < C)$

**using** *assms* **by** *auto*

**then show**  $C \in Rf\ N$

**unfolding** *Rf\_def* **by** *blast*

**qed**

**lemma** *contradiction\_Rf*:  $\{\#\} \in N \Longrightarrow Rf\ N = UNIV - \{\{\#\}\}$

**unfolding** *Rf\_def* **by** *force*

The following results correspond to Lemma 4.5. The lemma *wlog\_non\_Rf* generalizes the core of the argument.

**lemma** *Rf\_mono*:  $N \subseteq N' \Longrightarrow Rf\ N \subseteq Rf\ N'$

**unfolding** *Rf\_def* **by** *auto*

**lemma** *wlog\_non\_Rf*:

**assumes**  $ex: \exists DD. \text{set\_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)$

**shows**  $\exists DD. \text{set\_mset } DD \subseteq N - Rf\ N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)$

**proof** –

**from** *ex* **obtain** *DD0* **where**

$dd0: DD0 \in \{DD. \text{set\_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D)\}$

**by** *blast*

**have**  $\exists DD. \text{set\_mset } DD \subseteq N \wedge (\forall I. I \models_m DD + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD \longrightarrow D' < D) \wedge$   
 $(\forall DD'. \text{set\_mset } DD' \subseteq N \wedge (\forall I. I \models_m DD' + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD' \longrightarrow D' < D) \longrightarrow$   
 $DD \leq DD')$

**using** *wf\_eq\_minimal*[*THEN iffD1, rule\_format, OF wf\_less\_multiset dd0*]

**unfolding** *not\_le[symmetric]* **by** *blast*

**then obtain** *DD* **where**

*dd\_subs\_n*:  $\text{set\_mset } DD \subseteq N$  **and**

*ddcc\_imp\_e*:  $\forall I. I \models_m DD + CC \longrightarrow I \models E$  **and**

*dd\_lt\_d*:  $\forall D'. D' \in\# DD \longrightarrow D' < D$  **and**

*d\_min*:  $\forall DD'. \text{set\_mset } DD' \subseteq N \wedge (\forall I. I \models_m DD' + CC \longrightarrow I \models E) \wedge (\forall D'. D' \in\# DD' \longrightarrow D' < D) \longrightarrow$   
 $DD \leq DD'$

**by** *blast*

**have**  $\forall Da. Da \in\# DD \longrightarrow Da \notin Rf\ N$

**proof** *clarify*

**fix** *Da*

**assume**

*da\_in\_dd*:  $Da \in\# DD$  **and**

*da\_rf*:  $Da \in Rf\ N$

**from** *da\_rf* **obtain** *DD'* **where**

*dd'\_subs\_n*:  $\text{set\_mset } DD' \subseteq N$  **and**

*dd'\_imp\_da*:  $\forall I. I \models_m DD' \longrightarrow I \models Da$  **and**

*dd'\_lt\_da*:  $\forall D'. D' \in\# DD' \longrightarrow D' < Da$

**unfolding** *Rf\_def* **by** *blast*

**define** *DDa* **where**

$DDa = DD - \{\#Da\# \} + DD'$

**have**  $\text{set\_mset } DDa \subseteq N$

**unfolding** *DDa\_def* **using** *dd\_subs\_n dd'\_subs\_n*

```

    by (meson contra_subsetD in_diffD subsetI union_iff)
  moreover have  $\forall I. I \models_m DDa + CC \longrightarrow I \models E$ 
    using dd'_imp_da ddcc_imp_e da_in_dd unfolding DDa_def true_cls_mset_def
    by (metis in_remove1_mset_neq union_iff)
  moreover have  $\forall D'. D' \in\# DDa \longrightarrow D' < D$ 
    using dd_lt_d dd'_lt_da da_in_dd unfolding DDa_def
    by (metis insert_DiffM2 order.strict_trans union_iff)
  moreover have  $DDa < DD$ 
    unfolding DDa_def
    by (meson da_in_dd dd'_lt_da mset.lt.single_right_iff single_subset_iff union_le_diff_plus)
  ultimately show False
    using d_min unfolding less_eq_multiset_def by (auto intro!: antisym)
qed
then show ?thesis
  using dd_subs_n ddcc_imp_e dd_lt_d by auto
qed

```

```

lemma Rf_imp_ex_non_Rf:
  assumes  $C \in Rf\ N$ 
  shows  $\exists CC. set\_mset\ CC \subseteq N - Rf\ N \wedge (\forall I. I \models_m CC \longrightarrow I \models C) \wedge (\forall C'. C' \in\# CC \longrightarrow C' < C)$ 
  using assms by (auto simp: Rf_def intro: wlog_non_Rf[of - {#}], simplified)

```

```

lemma Rf_subs_Rf_diff_Rf:  $Rf\ N \subseteq Rf\ (N - Rf\ N)$ 
proof
  fix C
  assume c_rf:  $C \in Rf\ N$ 
  then obtain CC where
    cc_subs:  $set\_mset\ CC \subseteq N - Rf\ N$  and
    cc_imp_c:  $\forall I. I \models_m CC \longrightarrow I \models C$  and
    cc_lt_c:  $\forall C'. C' \in\# CC \longrightarrow C' < C$ 
    using Rf_imp_ex_non_Rf by blast
  have  $\forall D. D \in\# CC \longrightarrow D \notin Rf\ N$ 
    using cc_subs by (simp add: subset_iff)
  then have cc_nr:
     $\bigwedge C\ DD. C \in\# CC \implies set\_mset\ DD \subseteq N \implies \forall I. I \models_m DD \longrightarrow I \models C \implies \exists D. D \in\# DD \wedge \sim D < C$ 
    unfolding Rf_def by auto metis
  have  $set\_mset\ CC \subseteq N$ 
    using cc_subs by auto
  then have  $set\_mset\ CC \subseteq N - \{C. \exists DD. set\_mset\ DD \subseteq N \wedge (\forall I. I \models_m DD \longrightarrow I \models C) \wedge (\forall D. D \in\# DD \longrightarrow D < C)\}$ 
    using cc_nr by auto
  then show  $C \in Rf\ (N - Rf\ N)$ 
    using cc_imp_c cc_lt_c unfolding Rf_def by auto
qed

```

```

lemma Rf_eq_Rf_diff_Rf:  $Rf\ N = Rf\ (N - Rf\ N)$ 
  by (metis Diff_subset Rf_mono Rf_subs_Rf_diff_Rf subset_antisym)

```

The following results correspond to Lemma 4.6.

```

lemma Ri_mono:  $N \subseteq N' \implies Ri\ N \subseteq Ri\ N'$ 
  unfolding Ri_def by auto

```

```

lemma Ri_subs_Ri_diff_Rf:  $Ri\ N \subseteq Ri\ (N - Rf\ N)$ 
proof
  fix  $\gamma$ 
  assume  $\gamma_{ri}: \gamma \in Ri\ N$ 
  then obtain CC D E where  $\gamma: \gamma = Infer\ CC\ D\ E$ 
    by (cases  $\gamma$ )
  have cc:  $CC = side\_prems\_of\ \gamma$  and d:  $D = main\_prem\_of\ \gamma$  and e:  $E = concl\_of\ \gamma$ 
    unfolding  $\gamma$  by simp_all
  obtain DD where
    set_mset DD  $\subseteq N$  and  $\forall I. I \models_m DD + CC \longrightarrow I \models E$  and  $\forall C. C \in\# DD \longrightarrow C < D$ 
    using  $\gamma_{ri}$  unfolding Ri_def cc d e by blast

```

**then obtain**  $DD'$  **where**  
 $set.mset\ DD' \subseteq N - Rf\ N$  **and**  $\forall I. I \models_m DD' + CC \longrightarrow I \models E$  **and**  $\forall D'. D' \in \# DD' \longrightarrow D' < D$   
**using**  $wlog\_non\_Rf$  **by**  $atomize\_elim$   $blast$   
**then show**  $\gamma \in Ri\ (N - Rf\ N)$   
**using**  $\gamma\_ri$  **unfolding**  $Ri\_def\ d\ cc\ e$  **by**  $blast$   
**qed**

**lemma**  $Ri\_eq\_Ri\_diff\_Rf$ :  $Ri\ N = Ri\ (N - Rf\ N)$   
**by** ( $metis\ Diff\_subset\ Ri\_mono\ Ri\_subs\_Ri\_diff\_Rf\ subset\_antisym$ )

**lemma**  $Ri\_subset\_Gamma$ :  $Ri\ N \subseteq \Gamma$   
**unfolding**  $Ri\_def$  **by**  $blast$

**lemma**  $Rf\_indep$ :  $N' \subseteq Rf\ N \implies Rf\ N \subseteq Rf\ (N - N')$   
**by** ( $metis\ Diff\_cancel\ Diff\_eq\_empty\_iff\ Diff\_mono\ Rf\_eq\_Rf\_diff\_Rf\ Rf\_mono$ )

**lemma**  $Ri\_indep$ :  $N' \subseteq Rf\ N \implies Ri\ N \subseteq Ri\ (N - N')$   
**by** ( $metis\ Diff\_mono\ Ri\_eq\_Ri\_diff\_Rf\ Ri\_mono\ order\_refl$ )

**lemma**  $Rf\_model$ :  
**assumes**  $I \models_s N - Rf\ N$   
**shows**  $I \models_s N$   
**proof** –  
**have**  $I \models_s Rf\ (N - Rf\ N)$   
**unfolding**  $true\_clss\_def$   
**by** ( $subst\ Rf\_def, simp\ add: true\_cls\_mset\_def, metis\ assms\ subset\_eq\ true\_clss\_def$ )  
**then have**  $I \models_s Rf\ N$   
**using**  $Rf\_subs\_Rf\_diff\_Rf\ true\_clss\_mono$  **by**  $blast$   
**then show**  $?thesis$   
**using**  $assms$  **by** ( $metis\ Un\_Diff\_cancel\ true\_clss\_union$ )  
**qed**

**lemma**  $Rf\_sat$ :  $satisfiable\ (N - Rf\ N) \implies satisfiable\ N$   
**by** ( $metis\ Rf\_model$ )

The following corresponds to Theorem 4.7:

**sublocale**  $redundancy\_criterion\ \Gamma\ Rf\ Ri$   
**by**  $unfold\_locales\ (rule\ Ri\_subset\_Gamma, (elim\ Rf\_mono\ Ri\_mono\ Rf\_indep\ Ri\_indep\ Rf\_sat)+)$

**end**

**locale**  $standard\_redundancy\_criterion\_reductive =$   
 $standard\_redundancy\_criterion + reductive\_inference\_system$   
**begin**

The following corresponds to Theorem 4.8:

**lemma**  $Ri\_effective$ :  
**assumes**  
 $in\_gamma: \gamma \in \Gamma$  **and**  
 $concl\_of\_in\_n\_un\_rf\_n: concl\_of\ \gamma \in N \cup Rf\ N$   
**shows**  $\gamma \in Ri\ N$   
**proof** –  
**obtain**  $CC\ D\ E$  **where**  
 $\gamma: \gamma = Infer\ CC\ D\ E$   
**by** ( $cases\ \gamma$ )  
**then have**  $cc: CC = side\_prems\_of\ \gamma$  **and**  $d: D = main\_prem\_of\ \gamma$  **and**  $e: E = concl\_of\ \gamma$   
**unfolding**  $\gamma$  **by**  $simp\_all$   
**note**  $e\_in\_n\_un\_rf\_n = concl\_of\_in\_n\_un\_rf\_n[folded\ e]$   
  
**{**  
**assume**  $E \in N$   
**moreover have**  $E < D$   
**using**  $\Gamma\_reductive\ e\ d\ in\_gamma$  **by**  $auto$

```

ultimately have
  set.mset {#E#} ⊆ N and ∀ I. I ⊨m {#E#} + CC ⟶ I ⊨ E and ∀ D'. D' ∈# {#E#} ⟶ D' < D
  by simp.all
then have redundant_infer N γ
  using cc d e by blast
}
moreover
{
  assume E ∈ Rf N
  then obtain DD where
    dd_sset: set.mset DD ⊆ N and
    dd_imp_e: ∀ I. I ⊨m DD ⟶ I ⊨ E and
    dd_lt_e: ∀ C'. C' ∈# DD ⟶ C' < E
  unfolding Rf_def by blast
  from dd_lt_e have ∀ Da. Da ∈# DD ⟶ Da < D
  using d e in_γ Γ_reductive less_trans by blast
  then have redundant_infer N γ
  using dd_sset dd_imp_e cc d e by blast
}
ultimately show γ ∈ Ri N
  using in_γ e.in_n.un_rf_n unfolding Ri_def by blast
qed

```

```

sublocale effective_redundancy_criterion Γ Rf Ri
  unfolding effective_redundancy_criterion_def
  by (intro conjI redundancy_criterion_axioms, unfold_locales, rule Ri_effective)

```

```

lemma contradiction_Rf: {#} ∈ N ⟹ Ri N = Γ
  unfolding Ri_def using Γ_reductive le_multiset_empty_right
  by (force intro: exI[of - {#{#}#}] le_multiset_empty_left)

```

end

```

locale standard_redundancy_criterion_counterex_reducing =
  standard_redundancy_criterion + counterex_reducing_inference_system
begin

```

The following result corresponds to Theorem 4.9.

```

lemma saturated_upto_complete_if:

```

```

  assumes
    satur: saturated_upto N and
    unsat: ¬ satisfiable N
  shows {#} ∈ N
proof (rule ccontr)
  assume ec_ni_n: {#} ∉ N

```

```

  define M where
    M = N - Rf N

```

```

  have ec_ni_m: {#} ∉ M
  unfolding M_def using ec_ni_n by fast

```

```

  have L_of_M ⊨s M
proof (rule ccontr)
  assume ¬ L_of_M ⊨s M
  then obtain D where
    d_in_m: D ∈ M and
    d_cex: ¬ L_of_M ⊨ D and
    d_min: ⋀ C. C ∈ M ⟹ C < D ⟹ L_of_M ⊨ C
  using ex_min_counterex by meson
  then obtain γ CC E where
    γ: γ = Infer CC D E and
    cc_subs_m: set.mset CC ⊆ M and

```

```

cc_true:  $L \text{ of } M \models_m CC$  and
 $\gamma_{in}: \gamma \in \Gamma$  and
e_cex:  $\neg L \text{ of } M \models E$  and
e_lt_d:  $E < D$ 
using  $\Gamma_{counterex\_reducing}[OF \ ec\_ni\_m] \ not\_less$  by metis
have cc:  $CC = side\_prems\_of \ \gamma$  and  $d: D = main\_prem\_of \ \gamma$  and  $e: E = concl\_of \ \gamma$ 
unfolding  $\gamma$  by simp_all
have  $\gamma \in Ri \ N$ 
by (rule set_mp[OF satur[unfolding saturated_upto_def inferences_from_def infer_from_def]])
(simp add:  $\gamma_{in} \ d_{in\_m} \ cc\_subs\_m \ cc[symmetric] \ d[symmetric] \ M\_def[symmetric]$ )
then have  $\gamma \in Ri \ M$ 
unfolding  $M\_def$  using  $Ri\_indep$  by fast
then obtain  $DD$  where
dd_subs_m:  $set\_mset \ DD \subseteq M$  and
dd_cc_imp_d:  $\forall I. I \models_m DD + CC \longrightarrow I \models E$  and
dd_lt_d:  $\forall C. C \in \# \ DD \longrightarrow C < D$ 
unfolding  $Ri\_def \ cc \ d \ e$  by blast
from dd_subs_m dd_lt_d have  $L \text{ of } M \models_m DD$ 
using d_min unfolding true_cls_mset_def by (metis contra_subsetD)
then have  $L \text{ of } M \models E$ 
using dd_cc_imp_d cc_true by auto
then show False
using e_cex by auto
qed
then have  $L \text{ of } M \models_s N$ 
using  $M\_def \ Rf\_model$  by blast
then show False
using unsat by blast
qed

theorem saturated_upto_complete:
assumes saturated_upto  $N$ 
shows  $\neg \text{satisfiable } N \longleftrightarrow \{\#\} \in N$ 
using assms saturated_upto_complete_if_true_cls_def by auto

end

end

```

## 14 First-Order Ordered Resolution Calculus with Selection

**theory** *FO\_Ordered\_Resolution*

**imports** *Abstract\_Substitution Ordered\_Ground\_Resolution Standard\_Redundancy*

**begin**

This material is based on Section 4.3 (“A Simple Resolution Prover for First-Order Clauses”) of Bachmair and Ganzinger’s chapter. Specifically, it formalizes the ordered resolution calculus for first-order standard clauses presented in Figure 4 and its related lemmas and theorems, including soundness and Lemma 4.12 (the lifting lemma).

The following corresponds to pages 41–42 of Section 4.3, until Figure 5 and its explanation.

**locale** *FO\_resolution* = *mgu subst\_atm id\_subst comp\_subst atm\_of\_atms renamings\_apart mgu*

**for**

$subst\_atm :: 'a :: wellorder \Rightarrow 's \Rightarrow 'a$  and  
 $id\_subst :: 's$  and  
 $comp\_subst :: 's \Rightarrow 's \Rightarrow 's$  and  
 $renamings\_apart :: 'a \text{ literal multiset list} \Rightarrow 's \text{ list}$  and  
 $atm\_of\_atms :: 'a \text{ list} \Rightarrow 'a$  and  
 $mgu :: 'a \text{ set set} \Rightarrow 's \text{ option} +$

**fixes**

$less\_atm :: 'a \Rightarrow 'a \Rightarrow bool$

**assumes**

$less\_atm\_stable: less\_atm \ A \ B \Longrightarrow less\_atm \ (A \cdot a \ \sigma) \ (B \cdot a \ \sigma)$

begin

## 14.1 Library

**lemma** *Bex\_cartesian\_product*:  $(\exists xy \in A \times B. P \ xy) \equiv (\exists x \in A. \exists y \in B. P \ (x, y))$   
**by** *simp*

**lemma** *length\_sorted\_list\_of\_multiset*[*simp*]:  $\text{length} \ (\text{sorted\_list\_of\_multiset} \ A) = \text{size} \ A$   
**by** (*metis mset\_sorted\_list\_of\_multiset size\_mset*)

**lemma** *eql\_map\_neg\_lit\_eql\_atm*:  
**assumes** *map*  $(\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As$   
**shows**  $As' \cdot al \ \eta = As$   
**using** *assms* **by** (*induction As' arbitrary: As*) *auto*

**lemma** *instance\_list*:  
**assumes** *negs*  $(\text{mset} \ As) = SDA' \cdot \eta$   
**shows**  $\exists As'. \text{negs} \ (\text{mset} \ As') = SDA' \wedge As' \cdot al \ \eta = As$   
**proof** –  
**from** *assms* **have** *negL*:  $\forall L \in \# \ SDA'. \text{is\_neg} \ L$   
**using** *Melem\_subst\_cls subst\_lit\_in\_negs\_is\_neg* **by** *metis*  
  
**from** *assms* **have**  $\{\#L \cdot l \ \eta. L \in \# \ SDA'\} = \text{mset} \ (\text{map} \ \text{Neg} \ As)$   
**using** *subst\_cls\_def* **by** *auto*  
**then** **have**  $\exists NAs'. \text{map} \ (\lambda L. L \cdot l \ \eta) \ NAs' = \text{map} \ \text{Neg} \ As \wedge \text{mset} \ NAs' = SDA'$   
**using** *image\_mset\_of\_subset\_list*[*of*  $\lambda L. L \cdot l \ \eta \ SDA' \text{map} \ \text{Neg} \ As$ ] **by** *auto*  
**then** **obtain**  $As' \text{ where } As'_p$ :  
 $\text{map} \ (\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As \wedge \text{mset} \ (\text{map} \ \text{Neg} \ As') = SDA'$   
**by** (*metis* (*no\_types*, *lifting*) *Neg\_atm\_of\_iff negL ex\_map\_conv set\_mset\_mset*)  
  
**have**  $\text{negs} \ (\text{mset} \ As') = SDA'$   
**using**  $As'_p$  **by** *auto*  
**moreover** **have**  $\text{map} \ (\lambda L. L \cdot l \ \eta) \ (\text{map} \ \text{Neg} \ As') = \text{map} \ \text{Neg} \ As$   
**using**  $As'_p$  **by** *auto*  
**then** **have**  $As' \cdot al \ \eta = As$   
**using** *eql\_map\_neg\_lit\_eql\_atm* **by** *auto*  
**ultimately** **show** *?thesis*  
**by** *blast*  
**qed**

## 14.2 First-Order Logic

**inductive** *true\_fo\_cls* ::  $'a \text{ interp} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$  (*infix*  $\models_{fo}$  50) **where**  
*true\_fo\_cls*:  $(\bigwedge \sigma. \text{is\_ground\_subst} \ \sigma \Longrightarrow I \models C \cdot \sigma) \Longrightarrow I \models_{fo} C$

**lemma** *true\_fo\_cls\_inst*:  $I \models_{fo} C \Longrightarrow \text{is\_ground\_subst} \ \sigma \Longrightarrow I \models C \cdot \sigma$   
**by** (*rule true\_fo\_cls.induct*)

**inductive** *true\_fo\_cls\_mset* ::  $'a \text{ interp} \Rightarrow 'a \text{ clause multiset} \Rightarrow \text{bool}$  (*infix*  $\models_{fom}$  50) **where**  
*true\_fo\_cls\_mset*:  $(\bigwedge \sigma. \text{is\_ground\_subst} \ \sigma \Longrightarrow I \models_m CC \cdot cm \ \sigma) \Longrightarrow I \models_{fom} CC$

**lemma** *true\_fo\_cls\_mset\_inst*:  $I \models_{fom} C \Longrightarrow \text{is\_ground\_subst} \ \sigma \Longrightarrow I \models_m C \cdot cm \ \sigma$   
**by** (*rule true\_fo\_cls\_mset.induct*)

**lemma** *true\_fo\_cls\_mset\_def2*:  $I \models_{fom} CC \longleftrightarrow (\forall C \in \# \ CC. I \models_{fo} C)$   
**unfolding** *true\_fo\_cls\_mset.simps true\_cls.simps true\_cls\_mset\_def* **by** *force*

**context**  
**fixes**  $S :: 'a \text{ clause} \Rightarrow 'a \text{ clause}$   
**begin**



### 14.3 Calculus

The following corresponds to Figure 4.

**definition** *maximal\_wrt* :: 'a  $\Rightarrow$  'a literal multiset  $\Rightarrow$  bool **where**  
*maximal\_wrt* A C  $\longleftrightarrow (\forall B \in \text{atms\_of } C. \neg \text{less\_atm } A B)$

**definition** *strictly\_maximal\_wrt* :: 'a  $\Rightarrow$  'a literal multiset  $\Rightarrow$  bool **where**  
*strictly\_maximal\_wrt* A C  $\equiv \forall B \in \text{atms\_of } C. A \neq B \wedge \neg \text{less\_atm } A B$

**lemma** *strictly\_maximal\_wrt\_maximal\_wrt*: *strictly\_maximal\_wrt* A C  $\implies$  *maximal\_wrt* A C  
**unfolding** *maximal\_wrt\_def* *strictly\_maximal\_wrt\_def* **by** auto

**inductive** *eligible* :: 's  $\Rightarrow$  'a list  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**

*eligible*:

$S \ DA = \text{negs } (\text{mset } As) \vee S \ DA = \{\#\} \wedge \text{length } As = 1 \wedge \text{maximal\_wrt } (As ! 0 \cdot a \ \sigma) (DA \cdot \sigma) \implies$   
*eligible*  $\sigma \ As \ DA$

**inductive**

*ord\_resolve*

:: 'a clause list  $\Rightarrow$  'a clause  $\Rightarrow$  'a multiset list  $\Rightarrow$  'a list  $\Rightarrow$  's  $\Rightarrow$  'a clause  $\Rightarrow$  bool

**where**

*ord\_resolve*:

$\text{length } CAs = n \implies$   
 $\text{length } Cs = n \implies$   
 $\text{length } AAs = n \implies$   
 $\text{length } As = n \implies$   
 $n \neq 0 \implies$   
 $(\forall i < n. CAs ! i = Cs ! i + \text{poss } (AAs ! i)) \implies$   
 $(\forall i < n. AAs ! i \neq \{\#\}) \implies$   
 $\text{Some } \sigma = \text{mgu } (\text{set\_mset } ' \text{ set } (\text{map2 } \text{add\_mset } As \ AAs)) \implies$   
 $\text{eligible } \sigma \ As \ (D + \text{negs } (\text{mset } As)) \implies$   
 $(\forall i < n. \text{strictly\_maximal\_wrt } (As ! i \cdot a \ \sigma) (Cs ! i \cdot \sigma)) \implies$   
 $(\forall i < n. S \ (CAs ! i) = \{\#\}) \implies$   
 $\text{ord\_resolve } CAs \ (D + \text{negs } (\text{mset } As)) \ AAs \ As \ \sigma \ (((\bigcup \# \text{mset } Cs) + D) \cdot \sigma)$

**inductive**

*ord\_resolve\_rename*

:: 'a clause list  $\Rightarrow$  'a clause  $\Rightarrow$  'a multiset list  $\Rightarrow$  'a list  $\Rightarrow$  's  $\Rightarrow$  'a clause  $\Rightarrow$  bool

**where**

*ord\_resolve\_rename*:

$\text{length } CAs = n \implies$   
 $\text{length } AAs = n \implies$   
 $\text{length } As = n \implies$   
 $(\forall i < n. \text{poss } (AAs ! i) \subseteq \# CAs ! i) \implies$   
 $\text{negs } (\text{mset } As) \subseteq \# DA \implies$   
 $\varrho = \text{hd } (\text{renamings\_apart } (DA \ \# \ CAs)) \implies$   
 $\varrho s = \text{tl } (\text{renamings\_apart } (DA \ \# \ CAs)) \implies$   
 $\text{ord\_resolve } (CAs \ \cdot\!\!\cdot\!\!\text{cl } \varrho s) (DA \cdot \varrho) (AAs \ \cdot\!\!\cdot\!\!\text{aml } \varrho s) (As \cdot \text{al } \varrho) \ \sigma \ E \implies$   
 $\text{ord\_resolve\_rename } CAs \ DA \ AAs \ As \ \sigma \ E$

**lemma** *ord\_resolve\_empty\_main\_prem*:  $\neg \text{ord\_resolve } Cs \ \{\#\} \ AAs \ As \ \sigma \ E$   
**by** (*simp* add: *ord\_resolve.simps*)

**lemma** *ord\_resolve\_rename\_empty\_main\_prem*:  $\neg \text{ord\_resolve\_rename } Cs \ \{\#\} \ AAs \ As \ \sigma \ E$   
**by** (*simp* add: *ord\_resolve\_empty\_main\_prem* *ord\_resolve\_rename.simps*)

### 14.4 Soundness

Soundness is not discussed in the chapter, but it is an important property. The following lemma is used to prove soundness. It is also used to prove Lemma 4.10, which is used to prove completeness.

**lemma** *ord\_resolve\_ground\_inst\_sound*:  
**assumes**

```

  res_e: ord_resolve CAs DA AAs As  $\sigma$  E and
  cc_inst_true:  $I \models m \text{ mset } CAs \cdot cm \sigma \cdot cm \eta$  and
  d_inst_true:  $I \models DA \cdot \sigma \cdot \eta$  and
  ground_subst_η: is_ground_subst  $\eta$ 
shows  $I \models E \cdot \eta$ 
using res_e
proof (cases rule: ord_resolve.cases)
  case (ord_resolve n Cs D)
  note da = this(1) and e = this(2) and cas_len = this(3) and cs_len = this(4) and
    aas_len = this(5) and as_len = this(6) and cas = this(8) and mgu = this(10) and
    len = this(1)

  have len: length CAs = length As
    using as_len cas_len by auto
  have is_ground_subst ( $\sigma \odot \eta$ )
    using ground_subst_η by (rule is_ground_comp_subst)
  then have cc_true:  $I \models m \text{ mset } CAs \cdot cm \sigma \cdot cm \eta$  and d_true:  $I \models DA \cdot \sigma \cdot \eta$ 
    using cc_inst_true d_inst_true by auto

  from mgu have unif:  $\forall i < n. \forall A \in \#AAs ! i. A \cdot a \sigma = As ! i \cdot a \sigma$ 
    using mgu_unifier as_len aas_len by blast

  show  $I \models E \cdot \eta$ 
  proof (cases  $\forall A \in \text{set } As. A \cdot a \sigma \cdot a \eta \in I$ )
    case True
    then have  $\neg I \models \text{negs } (mset As) \cdot \sigma \cdot \eta$ 
      unfolding true_cls_def[of I] by auto
    then have  $I \models D \cdot \sigma \cdot \eta$ 
      using d_true da by auto
    then show ?thesis
      unfolding e by auto
    next
    case False
    then obtain i where a_in_aa:  $i < \text{length } CAs$  and a_false:  $(As ! i) \cdot a \sigma \cdot a \eta \notin I$ 
      using da len by (metis in_set_conv_nth)
    define C where  $C \equiv Cs ! i$ 
    define BB where  $BB \equiv AAs ! i$ 
    have c_cf':  $C \subseteq \# \bigcup \# mset CAs$ 
      unfolding C_def using a_in_aa cas cas_len
      by (metis less_subset_eq_Union_mset mset_subset_eq_add_left subset_mset.order.trans)
    have c_in_cc:  $C + \text{poss } BB \in \# mset CAs$ 
      using C_def BB_def a_in_aa cas_len in_set_conv_nth cas by fastforce
    {
      fix B
      assume  $B \in \# BB$ 
      then have  $B \cdot a \sigma = (As ! i) \cdot a \sigma$ 
        using unif a_in_aa cas_len unfolding BB_def by auto
    }
    then have  $\neg I \models \text{poss } BB \cdot \sigma \cdot \eta$ 
      using a_false by (auto simp: true_cls_def)
    moreover have  $I \models (C + \text{poss } BB) \cdot \sigma \cdot \eta$ 
      using c_in_cc cc_true true_cls_mset_true_cls[of I mset CAs  $\cdot cm \sigma \cdot cm \eta$ ] by force
    ultimately have  $I \models C \cdot \sigma \cdot \eta$ 
      by simp
    then show ?thesis
      unfolding e subst_cls_union using c_cf' C_def a_in_aa cas_len cs_len
      by (metis no_types, lifting) mset_subset_eq_add_left nth_mem_mset set_mset_mono sum_mset.remove true_cls_mono
      subst_cls_mono
    qed
  qed

```

lemma ord\_resolve\_sound:

assumes

$res\_e: ord\_resolve\ CAs\ DA\ AAs\ As\ \sigma\ E$  and  
 $cc\_d.true: I \models_{fom} mset\ CAs + \{\#DA\# \}$   
**shows**  $I \models_{fo} E$   
**proof** (rule true\_fo\_cls, use res\_e in (cases rule: ord\_resolve.cases))  
**fix**  $\eta$   
**assume** ground\_subst\_η: is\_ground\_subst  $\eta$   
**case** (ord\_resolve n Cs D)  
**note**  $da = this(1)$  and  $e = this(2)$  and  $cas\_len = this(3)$  and  $cs\_len = this(4)$   
and  $aas\_len = this(5)$  and  $as\_len = this(6)$  and  $cas = this(8)$  and  $mgu = this(10)$   
  
**have** is\_ground\_subst  $(\sigma \odot \eta)$   
**using** ground\_subst\_η **by** (rule is\_ground\_comp\_subst)  
**then have** cas.true:  $I \models_m mset\ CAs \cdot cm\ \sigma \cdot cm\ \eta$  and  $da.true: I \models DA \cdot \sigma \cdot \eta$   
**using** true\_fo\_cls\_mset\_inst[OF cc\_d.true, of  $\sigma \odot \eta$ ] **by** auto  
**show**  $I \models E \cdot \eta$   
**using** ord\_resolve\_ground\_inst\_sound[OF res\_e cas.true da.true] ground\_subst\_η **by** auto  
**qed**

**lemma** subst\_sound:  $I \models_{fo} C \implies I \models_{fo} (C \cdot \varrho)$   
**by** (metis is\_ground\_comp\_subst subst\_cls\_comp\_subst true\_fo\_cls true\_fo\_cls\_inst)

**lemma** true\_fo\_cls\_mset\_true\_fo\_cls:  $I \models_{fom} CC \implies C \in \# CC \implies I \models_{fo} C$   
**using** true\_fo\_cls\_mset\_def2 **by** auto

**lemma** subst\_sound\_scl:  
**assumes**  
 $len: length\ P = length\ CAs$  and  
 $true\_cas: I \models_{fom} mset\ CAs$   
**shows**  $I \models_{fom} mset\ (CAs \cdot cl\ P)$   
**proof** –  
**from** true\_cas **have**  $\forall CA. CA \in \# mset\ CAs \implies I \models_{fo} CA$   
**using** true\_fo\_cls\_mset\_true\_fo\_cls **by** auto  
**then have**  $\forall i < length\ CAs. I \models_{fo} CAs ! i$   
**using** in\_set\_conv\_nth **by** auto  
**then have** true\_cp:  $\forall i < length\ CAs. I \models_{fo} CAs ! i \cdot P ! i$   
**using** subst\_sound len **by** auto  
  
{  
**fix**  $CA$   
**assume**  $CA \in \# mset\ (CAs \cdot cl\ P)$   
**then obtain**  $i$  **where**  
 $i.x: i < length\ (CAs \cdot cl\ P)$   $CA = (CAs \cdot cl\ P) ! i$   
**by** (metis in\_mset\_conv\_nth)  
**then have**  $I \models_{fo} CA$   
**using** true\_cp **unfolding** subst\_cls\_lists\_def **by** (simp add: len)  
}  
**then show** ?thesis  
**unfolding** true\_fo\_cls\_mset\_def2 **by** auto  
**qed**

This is a lemma needed to prove Lemma 4.11.

**lemma** ord\_resolve\_rename\_ground\_inst\_sound:  
**assumes**  
 $ord\_resolve\_rename\ CAs\ DA\ AAs\ As\ \sigma\ E$  and  
 $\varrho s = tl\ (renamings\_apart\ (DA\ \# CAs))$  and  
 $\varrho = hd\ (renamings\_apart\ (DA\ \# CAs))$  and  
 $I \models_m (mset\ (CAs \cdot cl\ \varrho s)) \cdot cm\ \sigma \cdot cm\ \eta$  and  
 $I \models DA \cdot \varrho \cdot \sigma \cdot \eta$  and  
is\_ground\_subst  $\eta$   
**shows**  $I \models E \cdot \eta$   
**using** assms **by** (cases rule: ord\_resolve\_rename.cases) (fast intro: ord\_resolve\_ground\_inst\_sound)

**lemma** ord\_resolve\_rename\_sound:

```

assumes
  res.e: ord_resolve_rename CAs DA AAs As  $\sigma$  E and
  cc_d.true: I  $\models_{\text{fom}}$  (mset CAs) + {#DA#}
shows I  $\models_{\text{fo}}$  E
using res.e
proof (cases rule: ord_resolve_rename.cases)
case (ord_resolve_rename n  $\varrho$   $\varrho$ s)
note  $\varrho$ s = this(7) and res = this(8)
have len: length  $\varrho$ s = length CAs
using  $\varrho$ s renames_apart by auto
have I  $\models_{\text{fom}}$  mset (CAs  $\cdot$ cl  $\varrho$ s) + {#DA  $\cdot$   $\varrho$ #}
using subst_sound_scl[OF len, of I] subst_sound cc_d.true by (simp add: true_fo_cls_mset_def2)
then show I  $\models_{\text{fo}}$  E
using ord_resolve_sound[OF res] by simp
qed

```

## 14.5 Other Basic Properties

**lemma** ord\_resolve\_unique:

```

assumes
  ord_resolve CAs DA AAs As  $\sigma$  E and
  ord_resolve CAs DA AAs As  $\sigma'$  E'
shows  $\sigma = \sigma' \wedge E = E'$ 
using assms
proof (cases rule: ord_resolve.cases[case_product ord_resolve.cases], intro conjI)
case (ord_resolve_ord_resolve CAs n Cs AAs As  $\sigma''$  DA CAs' n' Cs' AAs' As'  $\sigma'''$  DA')
note res = this(1-17) and res' = this(18-34)

show  $\sigma = \sigma'$ 
using res(3-5,14) res'(3-5,14) by (metis option.inject)

have Cs = Cs'
using res(1,3,7,8,12) res'(1,3,7,8,12) by (metis add_right_imp_eq nth_equalityI)
moreover have DA = DA'
using res(2,4) res'(2,4) by fastforce
ultimately show E = E'
using res(5,6) res'(5,6)  $\sigma$  by blast
qed

```

**lemma** ord\_resolve\_rename\_unique:

```

assumes
  ord_resolve_rename CAs DA AAs As  $\sigma$  E and
  ord_resolve_rename CAs DA AAs As  $\sigma'$  E'
shows  $\sigma = \sigma' \wedge E = E'$ 
using assms unfolding ord_resolve_rename.simps using ord_resolve_unique by meson

```

**lemma** ord\_resolve\_max\_side\_premis: ord\_resolve CAs DA AAs As  $\sigma$  E  $\implies$  length CAs  $\leq$  size DA  
**by** (auto elim!: ord\_resolve.cases)

**lemma** ord\_resolve\_rename\_max\_side\_premis:

```

  ord_resolve_rename CAs DA AAs As  $\sigma$  E  $\implies$  length CAs  $\leq$  size DA
by (elim ord_resolve_rename.cases, drule ord_resolve_max_side_premis, simp add: renames_apart)

```

## 14.6 Inference System

**definition** ord\_FO $\Gamma$  :: 'a inference set **where**

```

  ord_FO $\Gamma$  = {Infer (mset CAs) DA E | CAs DA AAs As  $\sigma$  E. ord_resolve_rename CAs DA AAs As  $\sigma$  E}

```

**interpretation** ord\_FO\_resolution: inference\_system ord\_FO $\Gamma$  .

**lemma** exists\_compose:  $\exists x. P(f x) \implies \exists y. P y$   
**by** meson

**lemma** finite\_ord\_FO\_resolution\_inferences\_between:

```

assumes fin_cc: finite CC
shows finite (ord.FO_resolution.inferences.between CC C)
proof –
  let ?CCC = CC  $\cup$   $\{C\}$ 

  define all_AA where all_AA =  $(\bigcup D \in ?CCC. \text{atms\_of } D)$ 
  define max_ary where max_ary = Max (size ‘ ?CCC)
  define CAS where CAS =  $\{CAs. CAs \in \text{lists } ?CCC \wedge \text{length } CAs \leq \text{max\_ary}\}$ 
  define AS where AS =  $\{As. As \in \text{lists } all\_AA \wedge \text{length } As \leq \text{max\_ary}\}$ 
  define AAS where AAS =  $\{AAs. AAs \in \text{lists } (mset ‘ AS) \wedge \text{length } AAs \leq \text{max\_ary}\}$ 

  note defs = all_AA_def max_ary_def CAS_def AS_def AAS_def

  let ?infer_of =
     $\lambda CAs DA AAs As. \text{Infer } (mset CAs) DA (THE E. \exists \sigma. \text{ord\_resolve\_rename } CAs DA AAs As \sigma E)$ 

  let ?Z =  $\{\gamma \mid CAs DA AAs As \sigma E \gamma. \gamma = \text{Infer } (mset CAs) DA E$ 
     $\wedge \text{ord\_resolve\_rename } CAs DA AAs As \sigma E \wedge \text{infer\_from } ?CCC \gamma \wedge C \in \# \text{prems\_of } \gamma\}$ 
  let ?Y =  $\{\text{Infer } (mset CAs) DA E \mid CAs DA AAs As \sigma E.$ 
     $\text{ord\_resolve\_rename } CAs DA AAs As \sigma E \wedge \text{set } CAs \cup \{DA\} \subseteq ?CCC\}$ 
  let ?X =  $\{?infer\_of CAs DA AAs As \mid CAs DA AAs As. CAs \in CAS \wedge DA \in ?CCC \wedge AAs \in AAS \wedge As \in AS\}$ 
  let ?W = CAS  $\times$  ?CCC  $\times$  AAS  $\times$  AS

  have fin_w: finite ?W
    unfolding defs using fin_cc by (simp add: finite_lists_length_le lists_eq_set)

  have ?Z  $\subseteq$  ?Y
    by (force simp: infer_from_def)
  also have  $\dots \subseteq$  ?X
  proof –
    {
      fix CAs DA AAs As  $\sigma E$ 
      assume
        res_e: ord_resolve_rename CAs DA AAs As  $\sigma E$  and
        da_in: DA  $\in$  ?CCC and
        cas_sub: set CAs  $\subseteq$  ?CCC

      have E = (THE E. \exists \sigma. ord_resolve_rename CAs DA AAs As  $\sigma E$ )
         $\wedge CAs \in CAS \wedge AAs \in AAS \wedge As \in AS$  (is ?e  $\wedge$  ?cas  $\wedge$  ?aas  $\wedge$  ?as)
      proof (intro conjI)
        show ?e
          using res_e ord_resolve_rename_unique by (blast intro: the_equality[symmetric])
      next
        show ?cas
          unfolding CAS_def max_ary_def using cas_sub
          ord_resolve_rename_max_side_premis[OF res_e] da_in fin_cc
          by (auto simp add: Max_ge_iff)
      next
        show ?aas
          using res_e
      proof (cases rule: ord_resolve_rename.cases)
        case (ord_resolve_rename n  $\varrho$  qs)
        note len_cas = this(1) and len_aas = this(2) and len_as = this(3) and
          aas_sub = this(4) and as_sub = this(5) and res_e' = this(8)

        show ?thesis
          unfolding AAS_def
        proof (clarify, intro conjI)
          show AAs  $\in$  lists (mset ‘ AS)
            unfolding AS_def image_def
          proof clarsimp
            fix AA
            assume AA  $\in$  set AAs

```

```

then obtain  $i$  where
   $i_{lt}$ :  $i < n$  and
   $aa$ :  $AA = AAs ! i$ 
  by (metis in_set_conv_nth len_aas)

have  $casi\_in$ :  $CAs ! i \in ?CCC$ 
  using  $i_{lt}$  len_cas cas_sub nth_mem by blast

have  $pos\_aa\_sub$ :  $poss\ AA \subseteq \# CAs ! i$ 
  using  $aa$  aas_sub  $i_{lt}$  by blast
then have  $set\_mset\ AA \subseteq atms\_of\ (CAs ! i)$ 
  by (metis atms_of_poss lits_subseteq_imp_atms_subseteq set_mset_mono)
also have  $aa\_sub$ :  $\dots \subseteq all\_AA$ 
  unfolding all_AA_def using  $casi\_in$  by force
finally have  $aa\_sub$ :  $set\_mset\ AA \subseteq all\_AA$ 
.

have  $size\ AA = size\ (poss\ AA)$ 
  by simp
also have  $\dots \leq size\ (CAs ! i)$ 
  by (rule size_mset_mono[OF pos_aa_sub])
also have  $\dots \leq max\_ary$ 
  unfolding max_ary_def using fin_cc  $casi\_in$  by auto
finally have  $sz\_aa$ :  $size\ AA \leq max\_ary$ 
.

let  $?As' = sorted\_list\_of\_multiset\ AA$ 

have  $?As' \in lists\ all\_AA$ 
  using  $aa\_sub$  by auto
moreover have  $length\ ?As' \leq max\_ary$ 
  using  $sz\_aa$  by simp
moreover have  $AA = mset\ ?As'$ 
  by simp
ultimately show  $\exists xa. xa \in lists\ all\_AA \wedge length\ xa \leq max\_ary \wedge AA = mset\ xa$ 
  by blast
qed
next
have  $length\ AAs = length\ As$ 
  unfolding len_aas len_as ..
also have  $\dots \leq size\ DA$ 
  using  $as\_sub$  size_mset_mono by fastforce
also have  $\dots \leq max\_ary$ 
  unfolding max_ary_def using fin_cc  $da\_in$  by auto
finally show  $length\ AAs \leq max\_ary$ 
.
qed
qed
next
show  $?as$ 
  unfolding AS_def
proof (clarify, intro conjI)
  have  $set\ As \subseteq atms\_of\ DA$ 
    using  $res\_e[simplified\ ord\_resolve\_rename.simps]$ 
    by (metis atms_of_negs lits_subseteq_imp_atms_subseteq set_mset_mono set_mset_mset)
  also have  $\dots \subseteq all\_AA$ 
    unfolding all_AA_def using  $da\_in$  by blast
  finally show  $As \in lists\ all\_AA$ 
    unfolding lists_eq_set by simp
next
have  $length\ As \leq size\ DA$ 
  using  $res\_e[simplified\ ord\_resolve\_rename.simps]$ 
  ord_resolve_rename_max_side_premis[OF  $res\_e$ ] by auto

```

```

    also have  $\text{size } DA \leq \text{max\_ary}$ 
    unfolding  $\text{max\_ary\_def}$  using  $\text{fin\_cc } \text{da\_in}$  by auto
    finally show  $\text{length } As \leq \text{max\_ary}$ 
  }
qed
qed
}
then show ?thesis
  by simp fast
qed
also have  $\dots \subseteq (\lambda(CAs, DA, AAs, As). ?infer\_of \ CAs \ DA \ AAs \ As) \ ' \ ?W$ 
  unfolding  $\text{image\_def } \text{Bex\_cartesian\_product}$  by fast
finally show ?thesis
  unfolding  $\text{inference\_system.inferences\_between\_def } \text{ord\_FO\_}\Gamma\text{\_def } \text{mem\_Collect\_eq}$ 
  by (fast intro:  $\text{rev\_finite\_subset}[OF \ \text{finite\_imageI}[OF \ \text{fin\_w}]])$ 
qed

lemma  $\text{ord\_FO\_resolution.inferences\_between\_empty\_empty}$ :
 $\text{ord\_FO\_resolution.inferences\_between } \{\} \ \{\#\} = \{\}$ 
  unfolding  $\text{ord\_FO\_resolution.inferences\_between\_def } \text{inference\_system.inferences\_between\_def}$ 
  infer\_from\_def  $\text{ord\_FO\_}\Gamma\text{\_def}$ 
  using  $\text{ord\_resolve.rename\_empty\_main\_prem}$  by auto

```

## 14.7 Lifting

The following corresponds to the passage between Lemmas 4.11 and 4.12.

```

context
  fixes  $M :: 'a \text{ clause set}$ 
  assumes  $\text{select}: \text{selection } S$ 
begin

interpretation  $\text{selection}$ 
  by (rule  $\text{select}$ )

definition  $S\_M :: 'a \text{ literal multiset} \Rightarrow 'a \text{ literal multiset}$  where
 $S\_M \ C =$ 
  (if  $C \in \text{grounding\_of\_clss } M$  then
    ( $\text{SOME } C'. \exists D \ \sigma. D \in M \wedge C = D \cdot \sigma \wedge C' = S \ D \cdot \sigma \wedge \text{is\_ground\_subst } \sigma$ )
  else
     $S \ C$ )

lemma  $S\_M\_grounding\_of\_clss$ :
  assumes  $C \in \text{grounding\_of\_clss } M$ 
  obtains  $D \ \sigma$  where
     $D \in M \wedge C = D \cdot \sigma \wedge S\_M \ C = S \ D \cdot \sigma \wedge \text{is\_ground\_subst } \sigma$ 
proof (atomize_elim, unfold  $S\_M\_def$  eqTrueI[OF  $\text{assms}$ ] if_True, rule someI_ex)
  from  $\text{assms}$  show  $\exists C' \ D \ \sigma. D \in M \wedge C = D \cdot \sigma \wedge C' = S \ D \cdot \sigma \wedge \text{is\_ground\_subst } \sigma$ 
  by (auto simp:  $\text{grounding\_of\_clss\_def } \text{grounding\_of\_cls\_def}$ )
qed

lemma  $S\_M\_not\_grounding\_of\_clss$ :  $C \notin \text{grounding\_of\_clss } M \Longrightarrow S\_M \ C = S \ C$ 
  unfolding  $S\_M\_def$  by simp

lemma  $S\_M\_selects\_subsesteq$ :  $S\_M \ C \subseteq\# \ C$ 
  by (metis  $S\_M\_grounding\_of\_clss \ S\_M\_not\_grounding\_of\_clss \ S\_selects\_subsesteq \ \text{subst\_cls\_mono\_mset}$ )

lemma  $S\_M\_selects\_neg\_lits$ :  $L \in\# \ S\_M \ C \Longrightarrow \text{is\_neg } L$ 
  by (metis  $\text{Melem\_subst\_cls } S\_M\_grounding\_of\_clss \ S\_M\_not\_grounding\_of\_clss \ S\_selects\_neg\_lits \ \text{subst\_lit\_is\_neg}$ )

end

end

```

The following corresponds to Lemma 4.12:

```

lemma map2_add_mset_map:
  assumes length AAs' = n and length As' = n
  shows map2 add_mset (As' · al η) (AAs' · aml η) = map2 add_mset As' AAs' · aml η
  using assms
proof (induction n arbitrary: AAs' As')
  case (Suc n)
  then have map2 add_mset (tl (As' · al η)) (tl (AAs' · aml η)) = map2 add_mset (tl As') (tl AAs') · aml η
    by simp
  moreover
  have Succ: length (As' · al η) = Suc n length (AAs' · aml η) = Suc n
    using Suc(3) Suc(2) by auto
  then have length (tl (As' · al η)) = n length (tl (AAs' · aml η)) = n
    by auto
  then have length (map2 add_mset (tl (As' · al η)) (tl (AAs' · aml η))) = n
    length (map2 add_mset (tl As') (tl AAs') · aml η) = n
    using Suc(2,3) by auto
  ultimately have ∀ i < n. tl (map2 add_mset ( (As' · al η)) ((AAs' · aml η))) ! i =
    tl (map2 add_mset (As') (AAs') · aml η) ! i
    using Suc(2,3) Succ by (simp add: map2_tl map_tl subst_atm_mset_list_def del: subst_atm_list_tl)
  moreover have nn: length (map2 add_mset ((As' · al η)) ((AAs' · aml η))) = Suc n
    length (map2 add_mset (As') (AAs') · aml η) = Suc n
    using Succ Suc by auto
  ultimately have ∀ i. i < Suc n ⟶ i > 0 ⟶
    map2 add_mset (As' · al η) (AAs' · aml η) ! i = (map2 add_mset As' AAs' · aml η) ! i
    by (auto simp: subst_atm_mset_list_def gr0_conv_Suc subst_atm_mset_def)
  moreover have add_mset (hd As' · a η) (hd AAs' · am η) = add_mset (hd As') (hd AAs') · am η
    unfolding subst_atm_mset_def by auto
  then have (map2 add_mset (As' · al η) (AAs' · aml η)) ! 0 = (map2 add_mset (As') (AAs') · aml η) ! 0
    using Suc by (simp add: Succ(2) subst_atm_mset_def)
  ultimately have ∀ i < Suc n. (map2 add_mset (As' · al η) (AAs' · aml η)) ! i =
    (map2 add_mset (As') (AAs') · aml η) ! i
    using Suc by auto
  then show ?case
    using nn list_eq_iff_nth_eq by metis
qed auto

```

```

lemma maximal_wrt_subst: maximal_wrt (A · a σ) (C · σ) ⟹ maximal_wrt A C
  unfolding maximal_wrt_def using in_atms_of_subst less_atm_stable by blast

```

```

lemma strictly_maximal_wrt_subst: strictly_maximal_wrt (A · a σ) (C · σ) ⟹ strictly_maximal_wrt A C
  unfolding strictly_maximal_wrt_def using in_atms_of_subst less_atm_stable by blast

```

```

lemma ground_resolvent_subset:
  assumes
    gr_cas: is_ground_cls_list CAs and
    gr_da: is_ground_cls DA and
    res_e: ord_resolve S CAs DA AAs As σ E
  shows E ⊆ # (⋃ # mset CAs) + DA
  using res_e
proof (cases rule: ord_resolve.cases)
  case (ord_resolve n Cs D)
  note da = this(1) and e = this(2) and cas_len = this(3) and cs_len = this(4)
    and aas_len = this(5) and as_len = this(6) and cas = this(8) and mgu = this(10)
  then have cs_sub_cas: ⋃ # mset Cs ⊆ # ⋃ # mset CAs
    using subseqeq_list_Union_mset cas_len cs_len by force
  then have cs_sub_cas: ⋃ # mset Cs ⊆ # ⋃ # mset CAs
    using subseqeq_list_Union_mset cas_len cs_len by force
  then have gr_cs: is_ground_cls_list Cs
    using gr_cas by simp
  have d_sub_da: D ⊆ # DA
    by (simp add: da)
  then have gr_d: is_ground_cls D

```



```

using gr_da is_ground_cls_mono by auto

have is_ground_cls ( $\bigcup \# \text{ mset } Cs + D$ )
  using gr_cs gr_d by auto
with e have  $E = (\bigcup \# \text{ mset } Cs + D)$ 
  by auto
then show ?thesis
  using cs_sub_cas d_sub_da by (auto simp: subset_mset.add_mono)
qed

lemma ord_resolve_obtain_clauses:
  assumes
    res_e: ord_resolve ( $S\_M S M$ )  $CAs DA AAs As \sigma E$  and
    select: selection  $S$  and
    grounding:  $\{DA\} \cup \text{ set } CAs \subseteq \text{ grounding\_of\_clss } M$  and
    n: length  $CAs = n$  and
    d:  $DA = D + \text{ negs } (\text{ mset } As)$  and
    c:  $(\forall i < n. CAs ! i = Cs ! i + \text{ poss } (AAs ! i)) \text{ length } Cs = n \text{ length } AAs = n$ 
  obtains  $DA'' \eta'' CAs'' \eta s'' As'' AAs'' D'' Cs''$  where
    length  $CAs'' = n$ 
    length  $\eta s'' = n$ 
     $DA'' \in M$ 
     $DA'' \cdot \eta'' = DA$ 
     $S DA'' \cdot \eta'' = S\_M S M DA$ 
     $\forall CA'' \in \text{ set } CAs''. CA'' \in M$ 
     $CAs'' \cdot \text{ cl } \eta s'' = CAs$ 
     $\text{ map } S CAs'' \cdot \text{ cl } \eta s'' = \text{ map } (S\_M S M) CAs$ 
    is_ground_subst  $\eta''$ 
    is_ground_subst_list  $\eta s''$ 
     $As'' \cdot \text{ al } \eta'' = As$ 
     $AAs'' \cdot \text{ aml } \eta s'' = AAs$ 
    length  $As'' = n$ 
     $D'' \cdot \eta'' = D$ 
     $DA'' = D'' + (\text{ negs } (\text{ mset } As''))$ 
     $S\_M S M (D + \text{ negs } (\text{ mset } As)) \neq \{\#\} \implies \text{ negs } (\text{ mset } As'') = S DA''$ 
    length  $Cs'' = n$ 
     $Cs'' \cdot \text{ cl } \eta s'' = Cs$ 
     $\forall i < n. CAs'' ! i = Cs'' ! i + \text{ poss } (AAs'' ! i)$ 
    length  $AAs'' = n$ 
  using res_e
proof (cases rule: ord_resolve.cases)
  case (ord_resolve n_twin Cs_twins D_twin)
  note da = this(1) and e = this(2) and cas = this(8) and mgu = this(10) and eligible = this(11)
  from ord_resolve have n_twin = n D_twin = D
    using n d by auto
  moreover have Cs_twins = Cs
    using c cas n calculation(1) (length Cs_twins = n_twin) by (auto simp add: nth_equalityI)
  ultimately
  have nz:  $n \neq 0$  and cs_len: length  $Cs = n$  and aas_len: length  $AAs = n$  and as_len: length  $As = n$ 
    and da:  $DA = D + \text{ negs } (\text{ mset } As)$  and eligible: eligible  $(S\_M S M) \sigma As (D + \text{ negs } (\text{ mset } As))$ 
    and cas:  $\forall i < n. CAs ! i = Cs ! i + \text{ poss } (AAs ! i)$ 
    using ord_resolve by force+

  note  $n = \langle n \neq 0 \rangle \langle \text{ length } CAs = n \rangle \langle \text{ length } Cs = n \rangle \langle \text{ length } AAs = n \rangle \langle \text{ length } As = n \rangle$ 

  interpret  $S$ : selection  $S$  by (rule select)

  — Obtain FO side premises
  have  $\forall CA \in \text{ set } CAs. \exists CA'' \eta c''. CA'' \in M \wedge CA'' \cdot \eta c'' = CA \wedge S CA'' \cdot \eta c'' = S\_M S M CA \wedge \text{ is\_ground\_subst } \eta c''$ 
    using grounding  $S\_M\_grounding\_of\_clss$  select by (metis (no\_types) le\_supE subset\_iff)
  then have  $\forall i < n. \exists CA'' \eta c''. CA'' \in M \wedge CA'' \cdot \eta c'' = (CAs ! i) \wedge S CA'' \cdot \eta c'' = S\_M S M (CAs ! i) \wedge \text{ is\_ground\_subst } \eta c''$ 

```

```

using n by force
then obtain  $\eta s'' f$   $CAs'' f$  where  $f.p$ :
 $\forall i < n. CAs'' f i \in M$ 
 $\forall i < n. (CAs'' f i) \cdot (\eta s'' f i) = (CAs ! i)$ 
 $\forall i < n. S (CAs'' f i) \cdot (\eta s'' f i) = S\_M S M (CAs ! i)$ 
 $\forall i < n. is\_ground\_subst (\eta s'' f i)$ 
using n by (metis (no\_types))

```

```

define  $\eta s''$  where
 $\eta s'' = map \eta s'' f [0 ..<n]$ 
define  $CAs''$  where
 $CAs'' = map CAs'' f [0 ..<n]$ 

```

```

have length  $\eta s'' = n$  length  $CAs'' = n$ 
unfolding  $\eta s''\_def$   $CAs''\_def$  by auto
note  $n = \langle length \eta s'' = n \rangle \langle length CAs'' = n \rangle n$ 

```

— The properties we need of the FO side premises

```

have  $CAs''\_in\_M$ :  $\forall CA'' \in set CAs'', CA'' \in M$ 
unfolding  $CAs''\_def$  using  $f.p(1)$  by auto
have  $CAs''\_to\_CAs$ :  $CAs'' \cdot cl \eta s'' = CAs$ 
unfolding  $CAs''\_def$   $\eta s''\_def$  using  $f.p(2)$  by (auto simp: n intro: nth_equalityI)
have  $SCAs''\_to\_SMCAs$ :  $(map S CAs'') \cdot cl \eta s'' = map (S\_M S M) CAs$ 
unfolding  $CAs''\_def$   $\eta s''\_def$  using  $f.p(3)$  n by (force intro: nth_equalityI)
have  $sub\_ground$ :  $\forall \eta c'' \in set \eta s'', is\_ground\_subst \eta c''$ 
unfolding  $\eta s''\_def$  using  $f.p n$  by force
then have  $is\_ground\_subst\_list \eta s''$ 
using n unfolding  $is\_ground\_subst\_list\_def$  by auto

```

— Split side premises  $CAs''$  into  $Cs''$  and  $AA''$

```

obtain  $AA'' Cs''$  where  $AA'' \cdot Cs''_p$ :
 $AA'' \cdot aml \eta s'' = AA's length Cs'' = n Cs'' \cdot cl \eta s'' = Cs$ 
 $\forall i < n. CAs'' ! i = Cs'' ! i + poss (AA'' ! i) length AA'' = n$ 
proof —
have  $\forall i < n. \exists AA''. AA'' \cdot am \eta s'' ! i = AA's ! i \wedge poss AA'' \subseteq\# CAs'' ! i$ 
proof (rule, rule)
fix i
assume  $i < n$ 
have  $CAs'' ! i \cdot \eta s'' ! i = CAs ! i$ 
using  $\langle i < n \rangle \langle CAs'' \cdot cl \eta s'' = CAs \rangle n$  by force
moreover have  $poss (AA's ! i) \subseteq\# CAs ! i$ 
using  $\langle i < n \rangle cas$  by auto
ultimately obtain  $poss\_AA''$  where
 $nn: poss\_AA'' \cdot \eta s'' ! i = poss (AA's ! i) \wedge poss\_AA'' \subseteq\# CAs'' ! i$ 
using  $cas$   $image\_mset\_of\_subset$  unfolding  $subst\_cls\_def$  by metis
then have  $l: \forall L \in\# poss\_AA''. is\_pos L$ 
unfolding  $subst\_cls\_def$  by (metis Melem.subst_cls imageE literal.disc(1)
literal.map_disc_iff set_image_mset subst_cls_def subst_lit_def)

```

```

define  $AA''$  where
 $AA'' = image\_mset atm\_of poss\_AA''$ 

```

```

have  $na: poss AA'' = poss\_AA''$ 
using l unfolding  $AA''\_def$  by auto
then have  $AA'' \cdot am \eta s'' ! i = AA's ! i$ 
using nn by (metis (mono_tags) literal.inject(1) multiset.inj_map_strong subst_cls_poss)
moreover have  $poss AA'' \subseteq\# CAs'' ! i$ 
using na nn by auto
ultimately show  $\exists AA'. AA' \cdot am \eta s'' ! i = AA's ! i \wedge poss AA' \subseteq\# CAs'' ! i$ 
by blast
qed
then obtain  $AA's'' f$  where
 $AA's'' f.p: \forall i < n. AA's'' f i \cdot am \eta s'' ! i = AA's ! i \wedge (poss (AA's'' f i)) \subseteq\# CAs'' ! i$ 

```

```

by metis

define AAs'' where AAs'' = map AAs''f [0 ..<n]

then have length AAs'' = n
  by auto
note n = n ⟨length AAs'' = n⟩

from AAs''_def have ∀ i < n. AAs'' ! i · am ηs'' ! i = AAs ! i
  using AAs''f-p by auto
then have AAs'_AAs: AAs'' · aml ηs'' = AAs
  using n by (auto intro: nth_equalityI)

from AAs''_def have AAs''_in_CAs'': ∀ i < n. poss (AAs'' ! i) ⊆# CAs'' ! i
  using AAs''f-p by auto

define Cs'' where
  Cs'' = map2 (op -) CAs'' (map poss AAs'')

have length Cs'' = n
  using Cs''_def n by auto
note n = n ⟨length Cs'' = n⟩

have ∀ i < n. CAs'' ! i = Cs'' ! i + poss (AAs'' ! i)
  using AAs''_in_CAs'' Cs''_def n by auto
then have Cs'' · cl ηs'' = Cs
  using ⟨CAs'' · cl ηs'' = CAs⟩ AAs'_AAs cas n by (auto intro: nth_equalityI)

show ?thesis
  using that
    ⟨AAs'' · aml ηs'' = AAs⟩ ⟨Cs'' · cl ηs'' = Cs⟩ ⟨∀ i < n. CAs'' ! i = Cs'' ! i + poss (AAs'' ! i)⟩
    ⟨length AAs'' = n⟩ ⟨length Cs'' = n⟩
  by blast
qed

— Obtain FO main premise
have ∃ DA'' η''. DA'' ∈ M ∧ DA = DA'' · η'' ∧ S DA'' · η'' = S_M S M DA ∧ is_ground_subst η''
  using grounding S_M_grounding_of_cls select by (metis le_supE singletonI subsetCE)
then obtain DA'' η'' where
  DA''_η''_p: DA'' ∈ M ∧ DA = DA'' · η'' ∧ S DA'' · η'' = S_M S M DA ∧ is_ground_subst η''
  by auto
— The properties we need of the FO main premise
have DA''_in_M: DA'' ∈ M
  using DA''_η''_p by auto
have DA''_to_DA: DA'' · η'' = DA
  using DA''_η''_p by auto
have SDA''_to_SMDA: S DA'' · η'' = S_M S M DA
  using DA''_η''_p by auto
have is_ground_subst η''
  using DA''_η''_p by auto

— Split main premise DA'' into D'' and As''
obtain D'' As'' where D''As''_p:
  As'' · al η'' = As length As'' = n D'' · η'' = D DA'' = D'' + (negs (mset As''))
  S_M S M (D + negs (mset As)) ≠ {#} ⇒ negs (mset As'') = S DA''
proof -
{
  assume a: S_M S M (D + negs (mset As)) = {#} ∧ length As = (Suc 0)
    ∧ maximal_wrt (As ! 0 · a σ) ((D + negs (mset As)) · σ)
  then have as: mset As = {#As ! 0#}
    by (auto intro: nth_equalityI)
  then have negs (mset As) = {#Neg (As ! 0)#}
    by (simp add: ⟨mset As = {#As ! 0#}⟩)

```

```

then have  $DA = D + \{\#Neg (As ! 0)\# \}$ 
  using da by auto
then obtain L where  $L \in\# DA'' \wedge L \cdot l \eta'' = Neg (As ! 0)$ 
  using  $DA''\_to\_DA$  by (metis Melem_subst_cls mset_subset_eq_add_right single_subset_iff)
then have  $Neg (atm\_of L) \in\# DA'' \wedge Neg (atm\_of L) \cdot l \eta'' = Neg (As ! 0)$ 
  by (metis Neg_atm_of_iff literal.sel(2) subst_lit_is_pos)
then have  $[atm\_of L] \cdot al \eta'' = As \wedge negs (mset [atm\_of L]) \subseteq\# DA''$ 
  using as subst_lit_def by auto
then have  $\exists As'. As' \cdot al \eta'' = As \wedge negs (mset As') \subseteq\# DA''$ 
   $\wedge (S\_M S M (D + negs (mset As)) \neq \{\#\} \longrightarrow negs (mset As') = S DA'')$ 
  using a by blast
}
moreover
{
  assume  $S\_M S M (D + negs (mset As)) = negs (mset As)$ 
  then have  $negs (mset As) = S DA'' \cdot \eta''$ 
    using da  $\langle S DA'' \cdot \eta'' = S\_M S M DA \rangle$  by auto
  then have  $\exists As'. negs (mset As') = S DA'' \wedge As' \cdot al \eta'' = As$ 
    using instance_list[of As S DA''  $\eta''$ ] S.S.selects_neg_lits by auto
  then have  $\exists As'. As' \cdot al \eta'' = As \wedge negs (mset As') \subseteq\# DA''$ 
     $\wedge (S\_M S M (D + negs (mset As)) \neq \{\#\} \longrightarrow negs (mset As') = S DA'')$ 
    using S.S.selects_subseteq by auto
}
ultimately have  $\exists As''. As'' \cdot al \eta'' = As \wedge (negs (mset As'')) \subseteq\# DA''$ 
   $\wedge (S\_M S M (D + negs (mset As)) \neq \{\#\} \longrightarrow negs (mset As'') = S DA'')$ 
  using eligible unfolding eligible.simps by auto
then obtain As'' where
   $As'_p: As'' \cdot al \eta'' = As \wedge negs (mset As'') \subseteq\# DA''$ 
   $\wedge (S\_M S M (D + negs (mset As)) \neq \{\#\} \longrightarrow negs (mset As'') = S DA'')$ 
  by blast
then have  $length As'' = n$ 
  using as_len by auto
note  $n = n$  this

have  $As'' \cdot al \eta'' = As$ 
  using As'_p by auto

define D'' where
   $D'' = DA'' - negs (mset As'')$ 
then have  $DA'' = D'' + negs (mset As'')$ 
  using As'_p by auto
then have  $D'' \cdot \eta'' = D$ 
  using  $DA''\_to\_DA$  da As'_p by auto

have  $S\_M S M (D + negs (mset As)) \neq \{\#\} \implies negs (mset As'') = S DA''$ 
  using As'_p by blast
then show ?thesis
  using that  $\langle As'' \cdot al \eta'' = As \rangle \langle D'' \cdot \eta'' = D \rangle \langle DA'' = D'' + (negs (mset As'')) \rangle \langle length As'' = n \rangle$ 
  by metis
qed

show ?thesis
  using that [OF  $n(2,1)$   $DA''\_in\_M DA''\_to\_DA SDA''\_to\_SMDA CAs''\_in\_M CAs''\_to\_CAs SCAs''\_to\_SMCA$ 
     $\langle is\_ground\_subst \eta'' \rangle \langle is\_ground\_subst\_list \eta s'' \rangle \langle As'' \cdot al \eta'' = As \rangle$ 
     $\langle AAs'' \cdot aml \eta s'' = AAs \rangle$ 
     $\langle length As'' = n \rangle$ 
     $\langle D'' \cdot \eta'' = D \rangle$ 
     $\langle DA'' = D'' + (negs (mset As'')) \rangle$ 
     $\langle S\_M S M (D + negs (mset As)) \neq \{\#\} \implies negs (mset As'') = S DA'' \rangle$ 
     $\langle length Cs'' = n \rangle$ 
     $\langle Cs'' \cdot cl \eta s'' = Cs \rangle$ 
     $\langle \forall i < n. CAs'' ! i = Cs'' ! i + poss (AAs'' ! i) \rangle$ 
     $\langle length AAs'' = n \rangle]$ 

```

by auto  
qed

lemma

assumes  $Pos\ A \in \# \ C$   
shows  $A \in atms\_of\ C$   
using *assms*  
by (*simp add: atm\_iff\_pos\_or\_neg\_lit*)

lemma *ord\_resolve\_rename\_lifting*:

assumes  
  *sel\_stable*:  $\bigwedge \varrho\ C. is\_renaming\ \varrho \implies S\ (C \cdot \varrho) = S\ C \cdot \varrho$  and  
  *res\_e*: *ord\_resolve* (*S\_M S M*) *CAs DA AAs As*  $\sigma\ E$  and  
  *select*: *selection S* and  
  *grounding*:  $\{DA\} \cup set\ CAs \subseteq grounding\_of\_clss\ M$   
obtains  $\eta s\ \eta\ \eta 2\ CAs''\ DA''\ AAs''\ As''\ E''\ \tau$  where  
  *is\_ground\_subst*  $\eta$   
  *is\_ground\_subst\_list*  $\eta s$   
  *is\_ground\_subst*  $\eta 2$   
  *ord\_resolve\_rename* *S CAs'' DA'' AAs'' As''*  $\tau\ E''$

$CAs'' \cdot cl\ \eta s = CAs\ DA'' \cdot \eta = DA\ E'' \cdot \eta 2 = E$   
 $\{DA''\} \cup set\ CAs'' \subseteq M$

using *res\_e*

proof (*cases rule: ord\_resolve.cases*)

case (*ord\_resolve n Cs D*)

note *da* = *this*(1) and *e* = *this*(2) and *cas\_len* = *this*(3) and *cs\_len* = *this*(4) and  
  *aas\_len* = *this*(5) and *as\_len* = *this*(6) and *nz* = *this*(7) and *cas* = *this*(8) and  
  *aas\_not\_empty* = *this*(9) and *mgu* = *this*(10) and *eligible* = *this*(11) and *str\_max* = *this*(12) and  
  *sel\_empty* = *this*(13)

have *sel\_ren\_list\_inv*:

$\bigwedge \varrho s\ Cs. length\ \varrho s = length\ Cs \implies is\_renaming\_list\ \varrho s \implies map\ S\ (Cs \cdot cl\ \varrho s) = map\ S\ Cs \cdot cl\ \varrho s$   
using *sel\_stable* unfolding *is\_renaming\_list\_def* by (*auto intro: nth\_equalityI*)

note  $n = \langle n \neq 0 \rangle \langle length\ CAs = n \rangle \langle length\ Cs = n \rangle \langle length\ AAs = n \rangle \langle length\ As = n \rangle$

interpret *S*: *selection S* by (*rule select*)

obtain  $DA''\ \eta''\ CAs''\ \eta s''\ As''\ AAs''\ D''\ Cs''$  where *as''*:

*length CAs''* = *n*  
*length*  $\eta s''$  = *n*  
 $DA'' \in M$   
 $DA'' \cdot \eta'' = DA$   
 $S\ DA'' \cdot \eta'' = S\_M\ S\ M\ DA$   
 $\forall CA'' \in set\ CAs''. CA'' \in M$   
 $CAs'' \cdot cl\ \eta s'' = CAs$   
 $map\ S\ CAs'' \cdot cl\ \eta s'' = map\ (S\_M\ S\ M)\ CAs$   
*is\_ground\_subst*  $\eta''$   
*is\_ground\_subst\_list*  $\eta s''$   
 $As'' \cdot al\ \eta'' = As$   
 $AAs'' \cdot aml\ \eta s'' = AAs$   
*length As''* = *n*  
 $D'' \cdot \eta'' = D$   
 $DA'' = D'' + (negs\ (mset\ As''))$   
 $S\_M\ S\ M\ (D + negs\ (mset\ As)) \neq \{\#\} \implies negs\ (mset\ As'') = S\ DA''$   
*length Cs''* = *n*  
 $Cs'' \cdot cl\ \eta s'' = Cs$   
 $\forall i < n. CAs'' ! i = Cs'' ! i + poss\ (AAs'' ! i)$   
*length AAs''* = *n*

using *ord\_resolve\_obtain\_clauses*[*of S M CAs DA, OF res\_e select grounding n(2) (DA = D + negs (mset As))*  
   $\langle \forall i < n. CAs ! i = Cs ! i + poss\ (AAs ! i) \rangle \langle length\ Cs = n \rangle \langle length\ AAs = n \rangle$ , *of thesis*] by *blast*

**note**  $n = \langle \text{length } CAs'' = n \rangle \langle \text{length } \eta s'' = n \rangle \langle \text{length } As'' = n \rangle \langle \text{length } AAs'' = n \rangle \langle \text{length } Cs'' = n \rangle n$

**have**  $\text{length } (\text{renamings\_apart } (DA'' \# CAs'')) = \text{Suc } n$   
**using**  $n$  **renames\\_apart** **by** *auto*

**note**  $n = \text{this } n$

**define**  $\varrho$  **where**  
 $\varrho = \text{hd } (\text{renamings\_apart } (DA'' \# CAs''))$   
**define**  $\varrho s$  **where**  
 $\varrho s = \text{tl } (\text{renamings\_apart } (DA'' \# CAs''))$   
**define**  $DA'$  **where**  
 $DA' = DA'' \cdot \varrho$   
**define**  $D'$  **where**  
 $D' = D'' \cdot \varrho$   
**define**  $As'$  **where**  
 $As' = As'' \cdot \text{al } \varrho$   
**define**  $CAs'$  **where**  
 $CAs' = CAs'' \cdot \text{cl } \varrho s$   
**define**  $Cs'$  **where**  
 $Cs' = Cs'' \cdot \text{cl } \varrho s$   
**define**  $AAs'$  **where**  
 $AAs' = AAs'' \cdot \text{aml } \varrho s$   
**define**  $\eta'$  **where**  
 $\eta' = \text{inv\_renaming } \varrho \odot \eta''$   
**define**  $\eta s'$  **where**  
 $\eta s' = \text{map inv\_renaming } \varrho s \odot \eta s''$

**have**  $\text{renames\_}DA'': \text{is\_renaming } \varrho$   
**using**  $\text{renames\_apart}$  **unfolding**  $\varrho\_def$   
**by**  $(\text{metis length\_greater\_0\_conv list.exhaust\_sel list.set\_intros}(1) \text{ list.simps}(3))$

**have**  $\text{renames\_}CAs'': \text{is\_renaming\_list } \varrho s$   
**using**  $\text{renames\_apart}$  **unfolding**  $\varrho s\_def$   
**by**  $(\text{metis is\_renaming\_list\_def length\_greater\_0\_conv list.set\_sel}(2) \text{ list.simps}(3))$

**have**  $\text{length } \varrho s = n$   
**unfolding**  $\varrho s\_def$  **using**  $n$  **by** *auto*  
**note**  $n = n \langle \text{length } \varrho s = n \rangle$   
**have**  $\text{length } As' = n$   
**unfolding**  $As'\_def$  **using**  $n$  **by** *auto*  
**have**  $\text{length } CAs' = n$   
**using**  $as''(1)$   $n$  **unfolding**  $CAs'\_def$  **by** *auto*  
**have**  $\text{length } Cs' = n$   
**unfolding**  $Cs'\_def$  **using**  $n$  **by** *auto*  
**have**  $\text{length } AAs' = n$   
**unfolding**  $AAs'\_def$  **using**  $n$  **by** *auto*  
**have**  $\text{length } \eta s' = n$   
**using**  $as''(2)$   $n$  **unfolding**  $\eta s'\_def$  **by** *auto*  
**note**  $n = \langle \text{length } CAs' = n \rangle \langle \text{length } \eta s' = n \rangle \langle \text{length } As' = n \rangle \langle \text{length } AAs' = n \rangle \langle \text{length } Cs' = n \rangle n$

**have**  $DA'_DA: DA' \cdot \eta' = DA$   
**using**  $as''(4)$  **unfolding**  $\eta'\_def$   $DA'\_def$  **using**  $\text{renames\_}DA''$  **by** *simp*  
**have**  $D'_D: D' \cdot \eta' = D$   
**using**  $as''(14)$  **unfolding**  $\eta'\_def$   $D'\_def$  **using**  $\text{renames\_}DA''$  **by** *simp*  
**have**  $As'_As: As' \cdot \text{al } \eta' = As$   
**using**  $as''(11)$  **unfolding**  $\eta'\_def$   $As'\_def$  **using**  $\text{renames\_}DA''$  **by** *auto*  
**have**  $S DA' \cdot \eta' = S\_M S M DA$   
**using**  $as''(5)$  **unfolding**  $\eta'\_def$   $DA'\_def$  **using**  $\text{renames\_}DA''$  **sel\\_stable** **by** *auto*  
**have**  $CAs'\_CAs: CAs' \cdot \text{cl } \eta s' = CAs$   
**using**  $as''(7)$  **unfolding**  $CAs'\_def$   $\eta s'\_def$  **using**  $\text{renames\_apart}$   $\text{renames\_}CAs''$   $n$  **by** *auto*  
**have**  $Cs'\_Cs: Cs' \cdot \text{cl } \eta s' = Cs$   
**using**  $as''(18)$  **unfolding**  $Cs'\_def$   $\eta s'\_def$  **using**  $\text{renames\_apart}$   $\text{renames\_}CAs''$   $n$  **by** *auto*

```

have AAs'_AAs: AAs' ..aml  $\eta s' = AAs$ 
  using as''(12) unfolding  $\eta s'_def$  AAs'_def using renames_CAs'' using n by auto
have map S CAs' ..cl  $\eta s' = map (S.M S M) CAs$ 
  unfolding CAs'_def  $\eta s'_def$  using as''(8) n renames_CAs'' sel_ren_list_inv by auto

have DA'_split:  $DA' = D' + negs (mset As')$ 
  using as''(15) DA'_def D'_def As'_def by auto
then have D'_subset_DA':  $D' \subseteq \# DA'$ 
  by auto
from DA'_split have negs_As'_subset_DA':  $negs (mset As') \subseteq \# DA'$ 
  by auto

have CAs'_split:  $\forall i < n. CAs' ! i = Cs' ! i + poss (AAs' ! i)$ 
  using as''(19) CAs'_def Cs'_def AAs'_def n by auto
then have  $\forall i < n. Cs' ! i \subseteq \# CAs' ! i$ 
  by auto
from CAs'_split have poss_AAs'_subset_CAs':  $\forall i < n. poss (AAs' ! i) \subseteq \# CAs' ! i$ 
  by auto
then have AAs'_in_atms_of_CAs':  $\forall i < n. \forall A \in \# AAs' ! i. A \in atms\_of (CAs' ! i)$ 
  by (auto simp add: atm_iff_pos_or_neg_lit)

have as':
   $S.M S M (D + negs (mset As)) \neq \{\#\} \implies negs (mset As') = S DA'$ 
proof -
  assume a:  $S.M S M (D + negs (mset As)) \neq \{\#\}$ 
  then have  $negs (mset As') \cdot \varrho = S DA'' \cdot \varrho$ 
    using as''(16) unfolding  $\varrho\_def$  by metis
  then show  $negs (mset As') = S DA'$ 
    using As'_def DA'_def using sel_stable[of  $\varrho DA''$ ] renames_DA'' by auto
qed

have vd: var_disjoint ( $DA' \# CAs'$ )
  unfolding DA'_def CAs'_def using renames_apart[of  $DA'' \# CAs'$ ]
  unfolding  $\varrho\_def$   $\varrho s\_def$ 
  by (metis length_greater_0_conv list.exhaust_sel n(6) substitution.subst_cls_lists_Cons
    substitution.axioms zero_less_Suc)

— Introduce ground substitution
from vd DA'_DA CAs'_CAs have  $\exists \eta. \forall i < Suc\ n. \forall S. S \subseteq \# (DA' \# CAs') ! i \longrightarrow S \cdot (\eta' \# \eta s') ! i = S \cdot \eta$ 
  unfolding var_disjoint_def using n by auto
then obtain  $\eta$  where  $\eta\_p: \forall i < Suc\ n. \forall S. S \subseteq \# (DA' \# CAs') ! i \longrightarrow S \cdot (\eta' \# \eta s') ! i = S \cdot \eta$ 
  by auto
have  $\eta\_p\_lit: \forall i < Suc\ n. \forall L. L \in \# (DA' \# CAs') ! i \longrightarrow L \cdot l (\eta' \# \eta s') ! i = L \cdot l \eta$ 
proof (rule, rule, rule, rule)
  fix i :: nat and L :: 'a literal
  assume a:
     $i < Suc\ n$ 
     $L \in \# (DA' \# CAs') ! i$ 
  then have  $\forall S. S \subseteq \# (DA' \# CAs') ! i \longrightarrow S \cdot (\eta' \# \eta s') ! i = S \cdot \eta$ 
    using  $\eta\_p$  by auto
  then have  $\{\# L \#\} \cdot (\eta' \# \eta s') ! i = \{\# L \#\} \cdot \eta$ 
    using a by (meson single_subset_iff)
  then show  $L \cdot l (\eta' \# \eta s') ! i = L \cdot l \eta$  by auto
qed
have  $\eta\_p\_atm: \forall i < Suc\ n. \forall A. A \in atms\_of ((DA' \# CAs') ! i) \longrightarrow A \cdot a (\eta' \# \eta s') ! i = A \cdot a \eta$ 
proof (rule, rule, rule, rule)
  fix i :: nat and A :: 'a
  assume a:
     $i < Suc\ n$ 
     $A \in atms\_of ((DA' \# CAs') ! i)$ 
  then obtain L where  $L\_p: atm\_of L = A \wedge L \in \# (DA' \# CAs') ! i$ 
    unfolding atm_of_def by auto
  then have  $L \cdot l (\eta' \# \eta s') ! i = L \cdot l \eta$ 

```

```

    using  $\eta\_p\_lit$   $a$  by auto
  then show  $A \cdot a (\eta' \# \eta s') ! i = A \cdot a \eta$ 
    using  $L\_p$  unfolding subst_lit_def by (cases  $L$ ) auto
qed

have  $DA'_{DA}: DA' \cdot \eta = DA$ 
  using  $DA'_{DA} \eta\_p$  by auto
have  $D' \cdot \eta = D$  using  $\eta\_p D'_D n D'_{subset\_DA'}$  by auto
have  $As' \cdot al \eta = As$ 
proof (rule nth_equalityI)
  show  $length (As' \cdot al \eta) = length As$ 
    using  $n$  by auto
next
  show  $\forall i < length (As' \cdot al \eta). (As' \cdot al \eta) ! i = As ! i$ 
proof (rule, rule)
  fix  $i :: nat$ 
  assume  $a: i < length (As' \cdot al \eta)$ 
  have  $A\_eq: \forall A. A \in atms\_of DA' \longrightarrow A \cdot a \eta' = A \cdot a \eta$ 
    using  $\eta\_p\_atm n$  by force
  have  $As' ! i \in atms\_of DA'$ 
    using  $negs\_As'_{subset\_DA'}$  unfolding atms_of_def
    using  $a n$  by force
  then have  $As' ! i \cdot a \eta' = As' ! i \cdot a \eta$ 
    using  $A\_eq$  by simp
  then show  $(As' \cdot al \eta) ! i = As ! i$ 
    using  $As'_{As} \langle length As' = n \rangle a$  by auto
qed
qed

have  $S DA' \cdot \eta = S\_M S M DA$ 
  using  $\langle S DA' \cdot \eta' = S\_M S M DA \rangle \eta\_p S.S\_selects\_subseteq$  by auto

from  $\eta\_p$  have  $\eta\_p\_CAs': \forall i < n. (CAs' ! i) \cdot (\eta s' ! i) = (CAs' ! i) \cdot \eta$ 
  using  $n$  by auto
then have  $CAs' \cdot cl \eta s' = CAs' \cdot cl \eta$ 
  using  $n$  by (auto intro: nth_equalityI)
then have  $CAs'_{\eta\_fo\_CAs}: CAs' \cdot cl \eta = CAs$ 
  using  $CAs'_{CAs} \eta\_p n$  by auto

from  $\eta\_p$  have  $\forall i < n. S (CAs' ! i) \cdot \eta s' ! i = S (CAs' ! i) \cdot \eta$ 
  using  $S.S\_selects\_subseteq n$  by auto
then have  $map S CAs' \cdot cl \eta s' = map S CAs' \cdot cl \eta$ 
  using  $n$  by (auto intro: nth_equalityI)
then have  $SCAs'_{\eta\_fo\_SMCAs}: map S CAs' \cdot cl \eta = map (S\_M S M) CAs$ 
  using  $\langle map S CAs' \cdot cl \eta s' = map (S\_M S M) CAs \rangle$  by auto

have  $Cs' \cdot cl \eta = Cs$ 
proof (rule nth_equalityI)
  show  $length (Cs' \cdot cl \eta) = length Cs$ 
    using  $n$  by auto
next
  show  $\forall i < length (Cs' \cdot cl \eta). (Cs' \cdot cl \eta) ! i = Cs ! i$ 
proof (rule, rule)
  fix  $i :: nat$ 
  assume  $i < length (Cs' \cdot cl \eta)$ 
  then have  $a: i < n$ 
    using  $n$  by force
  have  $(Cs' \cdot cl \eta s') ! i = Cs ! i$ 
    using  $Cs'_{Cs} a n$  by force
  moreover
  have  $\eta\_p\_CAs': \forall S. S \subseteq \# CAs' ! i \longrightarrow S \cdot \eta s' ! i = S \cdot \eta$ 
    using  $\eta\_p a$  by force
  have  $Cs' ! i \cdot \eta s' ! i = (Cs' \cdot cl \eta) ! i$ 

```



```

    using  $\eta\_p\_CAs'$   $\langle \forall i < n. Cs' ! i \subseteq \# CAs' ! i \rangle$  a n by force
  then have  $(Cs' \cdot cl \ \eta s') ! i = (Cs' \cdot cl \ \eta) ! i$ 
    using a n by force
  ultimately show  $(Cs' \cdot cl \ \eta) ! i = Cs ! i$ 
    by auto
  thm  $Cs'_Cs \ \eta\_p \ \langle \forall i < n. Cs' ! i \subseteq \# CAs' ! i \rangle$  a
qed
qed

have  $AAs'_AAs: AAs' \cdot aml \ \eta = AAs$ 
proof (rule nth_equalityI)
  show  $length \ (AAs' \cdot aml \ \eta) = length \ AAs$ 
    using n by auto
next
show  $\forall i < length \ (AAs' \cdot aml \ \eta). (AAs' \cdot aml \ \eta) ! i = AAs ! i$ 
proof (rule, rule)
  fix i :: nat
  assume a:  $i < length \ (AAs' \cdot aml \ \eta)$ 
  then have  $i < n$ 
    using n by force
  then have  $\forall A. A \in atms\_of \ ((DA' \# CAs') ! Suc \ i) \longrightarrow A \cdot a \ (\eta' \# \eta s') ! Suc \ i = A \cdot a \ \eta$ 
    using  $\eta\_p\_atm \ n$  by force
  then have  $A\_eq: \forall A. A \in atms\_of \ (CAs' ! i) \longrightarrow A \cdot a \ \eta s' ! i = A \cdot a \ \eta$ 
    by auto
  have  $AAs\_CAs': \forall A \in \# AAs' ! i. A \in atms\_of \ (CAs' ! i)$ 
    using  $AAs'_in\_atms\_of\_CAs'$  unfolding  $atms\_of\_def$ 
    using a n by force
  then have  $AAs' ! i \cdot am \ \eta s' ! i = AAs' ! i \cdot am \ \eta$ 
    unfolding  $subst\_atm\_mset\_def$  using  $A\_eq$  unfolding  $subst\_atm\_mset\_def$  by auto
  then show  $(AAs' \cdot aml \ \eta) ! i = AAs ! i$ 
    using  $AAs'_AAs \ \langle length \ AAs' = n \rangle \ \langle length \ \eta s' = n \rangle$  a by auto
qed
qed

— Obtain MGU and substitution
obtain  $\tau \ \varphi$  where  $\tau \varphi$ :
  Some  $\tau = mgu \ (set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs'))$ 
   $\tau \odot \varphi = \eta \odot \sigma$ 
proof —
  have  $uu: is\_unifiers \ \sigma \ (set\_mset \ 'set \ (map2 \ add\_mset \ (As' \cdot al \ \eta) \ (AAs' \cdot aml \ \eta)))$ 
    using  $mgu \ mgu\_sound \ is\_mgu\_def$  unfolding  $\langle AAs' \cdot aml \ \eta = AAs \rangle$  using  $\langle As' \cdot al \ \eta = As \rangle$  by auto
  have  $\eta \sigma uni: is\_unifiers \ (\eta \odot \sigma) \ (set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs'))$ 
proof —
  have  $set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs' \cdot aml \ \eta) =$ 
     $set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs') \cdot ass \ \eta$ 
    unfolding  $subst\_atmss\_def \ subst\_atm\_mset\_list\_def$  using  $subst\_atm\_mset\_def \ subst\_atms\_def$ 
    by  $(simp \ add: image\_image \ subst\_atm\_mset\_def \ subst\_atms\_def)$ 
  then have  $is\_unifiers \ \sigma \ (set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs') \cdot ass \ \eta)$ 
    using uu by  $(auto \ simp: n \ map2\_add\_mset\_map)$ 
  then show  $?thesis$ 
    using  $is\_unifiers\_comp$  by auto
qed
then obtain  $\tau$  where
   $\tau\_p: Some \ \tau = mgu \ (set\_mset \ 'set \ (map2 \ add\_mset \ As' \ AAs'))$ 
  using  $mgu\_complete$ 
by  $(metis \ (mono\_tags, \ hide\_lams) \ List.finite\_set \ finite\_imageI \ finite\_set\_mset \ image\_iff)$ 
moreover then obtain  $\varphi$  where  $\varphi\_p: \tau \odot \varphi = \eta \odot \sigma$ 
  by  $(metis \ (mono\_tags, \ hide\_lams) \ finite\_set \ \eta \sigma uni \ finite\_imageI \ finite\_set\_mset \ image\_iff$ 
     $mgu\_sound \ set\_mset\_mset \ substitution\_ops.is\_mgu\_def)$ 
ultimately show  $thesis$ 
  using that by auto
qed

```

— Lifting eligibility

**have** *eligible'*:  $\text{eligible } S \tau As' (D' + \text{negs } (\text{mset } As'))$

**proof** —

**have**  $S\_M S M (D + \text{negs } (\text{mset } As)) = \text{negs } (\text{mset } As) \vee S\_M S M (D + \text{negs } (\text{mset } As)) = \{\#\} \wedge$   
 $\text{length } As = 1 \wedge \text{maximal\_wrt } (As ! 0 \cdot a \sigma) ((D + \text{negs } (\text{mset } As)) \cdot \sigma)$

**using** *eligible* **unfolding** *eligible.simps* **by** *auto*

**then show** *?thesis*

**proof**

**assume**  $S\_M S M (D + \text{negs } (\text{mset } As)) = \text{negs } (\text{mset } As)$

**then have**  $S\_M S M (D + \text{negs } (\text{mset } As)) \neq \{\#\}$

**using** *n* **by** *force*

**then have**  $S (D' + \text{negs } (\text{mset } As')) = \text{negs } (\text{mset } As')$

**using** *as' DA'\_split* **by** *auto*

**then show** *?thesis*

**unfolding** *eligible.simps[simplified]* **by** *auto*

**next**

**assume** *asm*:  $S\_M S M (D + \text{negs } (\text{mset } As)) = \{\#\} \wedge \text{length } As = 1 \wedge$

$\text{maximal\_wrt } (As ! 0 \cdot a \sigma) ((D + \text{negs } (\text{mset } As)) \cdot \sigma)$

**then have**  $S (D' + \text{negs } (\text{mset } As')) = \{\#\}$

**using**  $\langle D' \cdot \eta = D \rangle [\text{symmetric}] \langle As' \cdot al \eta = As \rangle [\text{symmetric}] \langle S (DA') \cdot \eta = S\_M S M (DA) \rangle$

**da** *DA'\_split subst\_cls\_empty\_iff* **by** *metis*

**moreover from** *asm* **have** *l*:  $\text{length } As' = 1$

**using**  $\langle As' \cdot al \eta = As \rangle$  **by** *auto*

**moreover from** *asm* **have**  $\text{maximal\_wrt } (As' ! 0 \cdot a (\tau \odot \varphi)) ((D' + \text{negs } (\text{mset } As')) \cdot (\tau \odot \varphi))$

**using**  $\langle As' \cdot al \eta = As \rangle \langle D' \cdot \eta = D \rangle$  **using** *l*  $\tau \varphi$  **by** *auto*

**then have**  $\text{maximal\_wrt } (As' ! 0 \cdot a \tau \cdot a \varphi) ((D' + \text{negs } (\text{mset } As')) \cdot \tau \cdot \varphi)$

**by** *auto*

**then have**  $\text{maximal\_wrt } (As' ! 0 \cdot a \tau) ((D' + \text{negs } (\text{mset } As')) \cdot \tau)$

**using** *maximal\_wrt\_subst* **by** *blast*

**ultimately show** *?thesis*

**unfolding** *eligible.simps[simplified]* **by** *auto*

**qed**

**qed**

— Lifting maximality

**have** *maximality*:  $\forall i < n. \text{strictly\_maximal\_wrt } (As' ! i \cdot a \tau) (Cs' ! i \cdot \tau)$

**proof** —

**from** *str\_max* **have**  $\forall i < n. \text{strictly\_maximal\_wrt } ((As' \cdot al \eta) ! i \cdot a \sigma) ((Cs' \cdot cl \eta) ! i \cdot \sigma)$

**using**  $\langle As' \cdot al \eta = As \rangle \langle Cs' \cdot cl \eta = Cs \rangle$  **by** *simp*

**then have**  $\forall i < n. \text{strictly\_maximal\_wrt } (As' ! i \cdot a (\tau \odot \varphi)) (Cs' ! i \cdot (\tau \odot \varphi))$

**using** *n*  $\tau \varphi$  **by** *simp*

**then have**  $\forall i < n. \text{strictly\_maximal\_wrt } (As' ! i \cdot a \tau \cdot a \varphi) (Cs' ! i \cdot \tau \cdot \varphi)$

**by** *auto*

**then show**  $\forall i < n. \text{strictly\_maximal\_wrt } (As' ! i \cdot a \tau) (Cs' ! i \cdot \tau)$

**using** *strictly\_maximal\_wrt\_subst*  $\tau \varphi$  **by** *blast*

**qed**

— Lifting nothing being selected

**have** *nothing\_selected*:  $\forall i < n. S (CAs' ! i) = \{\#\}$

**proof** —

**have**  $\forall i < n. (\text{map } S CAs' \cdot cl \eta) ! i = \text{map } (S\_M S M) CAs ! i$

**by** (*simp add*:  $\langle \text{map } S CAs' \cdot cl \eta = \text{map } (S\_M S M) CAs \rangle$ )

**then have**  $\forall i < n. S (CAs' ! i) \cdot \eta = S\_M S M (CAs ! i)$

**using** *n* **by** *auto*

**then have**  $\forall i < n. S (CAs' ! i) \cdot \eta = \{\#\}$

**using** *sel\_empty*  $\langle \forall i < n. S (CAs' ! i) \cdot \eta = S\_M S M (CAs ! i) \rangle$  **by** *auto*

**then show**  $\forall i < n. S (CAs' ! i) = \{\#\}$

**using** *subst\_cls\_empty\_iff* **by** *blast*

**qed**

— Lifting AAs's non-emptiness

**have**  $\forall i < n. AAs' ! i \neq \{\#\}$

**using**  $n$  *aas\_not\_empty*  $\langle AAs' \cdot aml \eta = AAs \rangle$  **by** *auto*

— Resolve the lifted clauses

**define**  $E'$  **where**

$E' = ((\bigcup \# \text{ mset } Cs') + D') \cdot \tau$

**have**  $res.e'$ : *ord\_resolve*  $S$   $CAs'$   $DA'$   $AAs'$   $As'$   $\tau$   $E'$

**using** *ord\_resolve.intros*[*of*  $CAs'$   $n$   $Cs'$   $AAs'$   $As'$   $\tau$   $S$   $D'$ ,

$OF$   $\dots \dots \langle \forall i < n. AAs' ! i \neq \{\#\} \rangle \tau \varphi(1)$  *eligible'*

$\langle \forall i < n. \text{strictly\_maximal\_wrt } (As' ! i \cdot a \tau) (Cs' ! i \cdot \tau) \rangle \langle \forall i < n. S (CAs' ! i) = \{\#\} \rangle]$

**unfolding**  $E'_\text{def}$  **using**  $DA'_\text{split } n$   $\langle \forall i < n. CAs' ! i = Cs' ! i + \text{poss } (AAs' ! i) \rangle$  **by** *blast*

— Prove resolvent instantiates to ground resolvent

**have**  $e'\varphi e$ :  $E' \cdot \varphi = E$

**proof** —

**have**  $E' \cdot \varphi = ((\bigcup \# \text{ mset } Cs') + D') \cdot (\tau \odot \varphi)$

**unfolding**  $E'_\text{def}$  **by** *auto*

**also have**  $\dots = (\bigcup \# \text{ mset } Cs' + D') \cdot (\eta \odot \sigma)$

**using**  $\tau \varphi$  **by** *auto*

**also have**  $\dots = (\bigcup \# \text{ mset } Cs + D) \cdot \sigma$

**using**  $\langle Cs' \cdot cl \eta = Cs \rangle \langle D' \cdot \eta = D \rangle$  **by** *auto*

**also have**  $\dots = E$

**using**  $e$  **by** *auto*

**finally show**  $e'\varphi e$ :  $E' \cdot \varphi = E$

**qed**

— Replace  $\varphi$  with a true ground substitution

**obtain**  $\eta 2$  **where**

*ground\_η2*: *is\_ground\_subst*  $\eta 2$   $E' \cdot \eta 2 = E$

**proof** —

**have** *is\_ground\_cls\_list*  $CAs$  *is\_ground\_cls*  $DA$

**using** *grounding* *grounding\_ground* **unfolding** *is\_ground\_cls\_list\_def* **by** *auto*

**then have** *is\_ground\_cls*  $E$

**using**  $res.e$  *ground\_resolvent\_subset* **by** (*force intro*: *is\_ground\_cls\_mono*)

**then show** *thesis*

**using** *that*  $e'\varphi e$  *make\_ground\_subst* **by** *auto*

**qed**

— Wrap up the proof

**have** *ord\_resolve*  $S$   $(CAs'' \cdot cl \varrho s)$   $(DA'' \cdot \varrho)$   $(AAs'' \cdot aml \varrho s)$   $(As'' \cdot al \varrho)$   $\tau$   $E'$

**using**  $res.e'$   $As'_\text{def}$   $\varrho_\text{def}$   $AAs'_\text{def}$   $\varrho s_\text{def}$   $DA'_\text{def}$   $\varrho_\text{def}$   $CAs'_\text{def}$   $\varrho s_\text{def}$  **by** *simp*

**moreover have**  $\forall i < n. \text{poss } (AAs'' ! i) \subseteq \# CAs'' ! i$

**using**  $as''(19)$  **by** *auto*

**moreover have**  $\text{negs } (\text{mset } As'') \subseteq \# DA''$

**using**  $local.as''(15)$  **by** *auto*

**ultimately have** *ord\_resolve\_rename*  $S$   $CAs''$   $DA''$   $AAs''$   $As''$   $\tau$   $E'$

**using** *ord\_resolve\_rename*[*of*  $CAs''$   $n$   $AAs''$   $As''$   $DA''$   $\varrho$   $\varrho s$   $S$   $\tau$   $E'$ ]  $\varrho_\text{def}$   $\varrho s_\text{def}$   $n$  **by** *auto*

**then show** *thesis*

**using** *that*[*of*  $\eta''$   $\eta s''$   $\eta 2$   $CAs''$   $DA''$ ]  $\langle is\_ground\_subst \eta'' \rangle \langle is\_ground\_subst\_list \eta s'' \rangle$

$\langle is\_ground\_subst \eta 2 \rangle \langle CAs'' \cdot cl \eta s'' = CAs \rangle \langle DA'' \cdot \eta'' = DA \rangle \langle E' \cdot \eta 2 = E \rangle \langle DA'' \in M \rangle$

$\langle \forall CA \in \text{set } CAs''. CA \in M \rangle$  **by** *blast*

**qed**

**end**

**end**

## 15 An Ordered Resolution Prover for First-Order Clauses

**theory** *FO\_Ordered\_Resolution\_Prover*

**imports** *FO\_Ordered\_Resolution*

**begin**

This material is based on Section 4.3 (“A Simple Resolution Prover for First-Order Clauses”) of Bachmair and Ganzinger’s chapter. Specifically, it formalizes the RP prover defined in Figure 5 and its related lemmas and theorems, including Lemmas 4.10 and 4.11 and Theorem 4.13 (completeness).

**definition** *is\_least* :: (nat  $\Rightarrow$  bool)  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
*is\_least* *P n*  $\longleftrightarrow P\ n \wedge (\forall n' < n. \neg P\ n')$

**lemma** *least\_exists*: *P n*  $\Longrightarrow \exists n. is\_least\ P\ n$   
**using** *exists\_least\_iff* **unfolding** *is\_least\_def* **by** *auto*

The following corresponds to page 42 and 43 of Section 4.3, from the explanation of RP to Lemma 4.10.

**type-synonym** 'a state = 'a clause set  $\times$  'a clause set  $\times$  'a clause set

**locale** *FO\_resolution\_prover* =  
*FO\_resolution subst\_atm id\_subst comp\_subst renamings\_apart atm\_of\_atms mgu less\_atm* +  
*selection S*  
**for**  
*S* :: ('a :: wellorder) clause  $\Rightarrow$  'a clause **and**  
*subst\_atm* :: 'a  $\Rightarrow$  's  $\Rightarrow$  'a **and**  
*id\_subst* :: 's **and**  
*comp\_subst* :: 's  $\Rightarrow$  's  $\Rightarrow$  's **and**  
*renamings\_apart* :: 'a clause list  $\Rightarrow$  's list **and**  
*atm\_of\_atms* :: 'a list  $\Rightarrow$  'a **and**  
*mgu* :: 'a set set  $\Rightarrow$  's option **and**  
*less\_atm* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool +  
**assumes**  
*sel\_stable*:  $\bigwedge \varrho\ C. is\_renaming\ \varrho \Longrightarrow S\ (C \cdot \varrho) = S\ C \cdot \varrho$  **and**  
*less\_atm\_ground*: *is\_ground\_atm A*  $\Longrightarrow is\_ground\_atm\ B \Longrightarrow less\_atm\ A\ B \Longrightarrow A < B$

**begin**

**fun** *N\_of\_state* :: 'a state  $\Rightarrow$  'a clause set **where**  
*N\_of\_state* (*N*, *P*, *Q*) = *N*

**fun** *P\_of\_state* :: 'a state  $\Rightarrow$  'a clause set **where**  
*P\_of\_state* (*N*, *P*, *Q*) = *P*

*O* denotes relation composition in Isabelle, so the formalization uses *Q* instead.

**fun** *Q\_of\_state* :: 'a state  $\Rightarrow$  'a clause set **where**  
*Q\_of\_state* (*N*, *P*, *Q*) = *Q*

**definition** *clss\_of\_state* :: 'a state  $\Rightarrow$  'a clause set **where**  
*clss\_of\_state* *St* = *N\_of\_state St*  $\cup$  *P\_of\_state St*  $\cup$  *Q\_of\_state St*

**abbreviation** *grounding\_of\_state* :: 'a state  $\Rightarrow$  'a clause set **where**  
*grounding\_of\_state St*  $\equiv$  *grounding\_of\_clss (clss\_of\_state St)*

**interpretation** *ord\_FO\_resolution*: *inference\_system ord\_FO*  $\Gamma\ S$  .

The following inductive predicate formalizes the resolution prover in Figure 5.

**inductive** *RP* :: 'a state  $\Rightarrow$  'a state  $\Rightarrow$  bool (**infix**  $\rightsquigarrow$  50) **where**  
*tautology\_deletion*: *Neg A*  $\in \# C \Longrightarrow Pos A \in \# C \Longrightarrow (N \cup \{C\}, P, Q) \rightsquigarrow (N, P, Q)$   
*forward\_subsumption*: *D*  $\in P \cup Q \Longrightarrow subsumes\ D\ C \Longrightarrow (N \cup \{C\}, P, Q) \rightsquigarrow (N, P, Q)$   
*backward\_subsumption\_P*: *D*  $\in N \Longrightarrow strictly\_subsumes\ D\ C \Longrightarrow (N, P \cup \{C\}, Q) \rightsquigarrow (N, P, Q)$   
*backward\_subsumption\_Q*: *D*  $\in N \Longrightarrow strictly\_subsumes\ D\ C \Longrightarrow (N, P, Q \cup \{C\}) \rightsquigarrow (N, P, Q)$   
*forward\_reduction*: *D* +  $\{\#L'\# \} \in P \cup Q \Longrightarrow -\ L = L' \cdot l\ \sigma \Longrightarrow D \cdot \sigma \subseteq \# C \Longrightarrow$   
 $(N \cup \{C + \{\#L'\#\}\}, P, Q) \rightsquigarrow (N \cup \{C\}, P, Q)$   
*backward\_reduction\_P*: *D* +  $\{\#L'\# \} \in N \Longrightarrow -\ L = L' \cdot l\ \sigma \Longrightarrow D \cdot \sigma \subseteq \# C \Longrightarrow$   
 $(N, P \cup \{C + \{\#L'\#\}\}, Q) \rightsquigarrow (N, P \cup \{C\}, Q)$   
*backward\_reduction\_Q*: *D* +  $\{\#L'\# \} \in N \Longrightarrow -\ L = L' \cdot l\ \sigma \Longrightarrow D \cdot \sigma \subseteq \# C \Longrightarrow$   
 $(N, P, Q \cup \{C + \{\#L'\#\}\}) \rightsquigarrow (N, P \cup \{C\}, Q)$   
*clause\_processing*:  $(N \cup \{C\}, P, Q) \rightsquigarrow (N, P \cup \{C\}, Q)$

| *inference\_computation*:  $N = \text{concls\_of } (\text{ord\_FO\_resolution.inferences\_between } Q \ C) \implies$   
 $(\{\}, P \cup \{C\}, Q) \rightsquigarrow (N, P, Q \cup \{C\})$

**lemma** *final\_RP*:  $\neg (\{\}, \{\}, Q) \rightsquigarrow St$   
**by** (*auto elim*: *RP.cases*)

**definition** *Sup\_state* :: 'a state llist  $\Rightarrow$  'a state **where**  
*Sup\_state* *Sts* =  
 (*Sup\_llist* (*lmap* *N\_of\_state* *Sts*), *Sup\_llist* (*lmap* *P\_of\_state* *Sts*),  
*Sup\_llist* (*lmap* *Q\_of\_state* *Sts*))

**definition** *Liminf\_state* :: 'a state llist  $\Rightarrow$  'a state **where**  
*Liminf\_state* *Sts* =  
 (*Liminf\_llist* (*lmap* *N\_of\_state* *Sts*), *Liminf\_llist* (*lmap* *P\_of\_state* *Sts*),  
*Liminf\_llist* (*lmap* *Q\_of\_state* *Sts*))

**context**  
**fixes** *Sts Sts'* :: 'a state llist  
**assumes** *Sts*: *lfinite* *Sts* *lfinite* *Sts'*  $\neg$  *lnull* *Sts*  $\neg$  *lnull* *Sts'* *llast* *Sts'* = *llast* *Sts*  
**begin**

**lemma**  
*N\_of\_Liminf\_state\_fin*: *N\_of\_state* (*Liminf\_state* *Sts'*) = *N\_of\_state* (*Liminf\_state* *Sts*) **and**  
*P\_of\_Liminf\_state\_fin*: *P\_of\_state* (*Liminf\_state* *Sts'*) = *P\_of\_state* (*Liminf\_state* *Sts*) **and**  
*Q\_of\_Liminf\_state\_fin*: *Q\_of\_state* (*Liminf\_state* *Sts'*) = *Q\_of\_state* (*Liminf\_state* *Sts*)  
**using** *Sts* **by** (*simp\_all add*: *Liminf\_state\_def lfinite.Liminf\_llist llast\_lmap*)

**lemma** *Liminf\_state\_fin*: *Liminf\_state* *Sts'* = *Liminf\_state* *Sts*  
**using** *N\_of\_Liminf\_state\_fin* *P\_of\_Liminf\_state\_fin* *Q\_of\_Liminf\_state\_fin*  
**by** (*simp add*: *Liminf\_state\_def*)

**end**

**context**  
**fixes** *Sts Sts'* :: 'a state llist  
**assumes** *Sts*:  $\neg$  *lfinite* *Sts* *emb* *Sts Sts'*  
**begin**

**lemma**  
*N\_of\_Liminf\_state\_inf*: *N\_of\_state* (*Liminf\_state* *Sts'*)  $\subseteq$  *N\_of\_state* (*Liminf\_state* *Sts*) **and**  
*P\_of\_Liminf\_state\_inf*: *P\_of\_state* (*Liminf\_state* *Sts'*)  $\subseteq$  *P\_of\_state* (*Liminf\_state* *Sts*) **and**  
*Q\_of\_Liminf\_state\_inf*: *Q\_of\_state* (*Liminf\_state* *Sts'*)  $\subseteq$  *Q\_of\_state* (*Liminf\_state* *Sts*)  
**using** *Sts* **by** (*simp\_all add*: *Liminf\_state\_def emb.Liminf\_llist.infinite emb\_lmap*)

**lemma** *cls\_of\_Liminf\_state\_inf*:  
*cls\_of\_state* (*Liminf\_state* *Sts'*)  $\subseteq$  *cls\_of\_state* (*Liminf\_state* *Sts*)  
**unfolding** *cls\_of\_state\_def*  
**using** *N\_of\_Liminf\_state\_inf* *P\_of\_Liminf\_state\_inf* *Q\_of\_Liminf\_state\_inf* **by** *blast*

**end**

**definition** *fair\_state\_seq* :: 'a state llist  $\Rightarrow$  bool **where**  
*fair\_state\_seq* *Sts*  $\longleftrightarrow$  *N\_of\_state* (*Liminf\_state* *Sts*) =  $\{\}$   $\wedge$  *P\_of\_state* (*Liminf\_state* *Sts*) =  $\{\}$

The following formalizes Lemma 4.10.

**context**  
**fixes**  
*Sts* :: 'a state llist  
**assumes**  
*deriv*: *chain* (*op*  $\rightsquigarrow$ ) *Sts* **and**  
*empty\_Q0*: *Q\_of\_state* (*lhd* *Sts*) =  $\{\}$

**begin**

**lemmas** *lhd\_lmap\_Sts* = *llist.map\_sel*(1)[*OF chain\_not\_innull*[*OF deriv*]]

**definition** *S\_Q* :: 'a clause  $\Rightarrow$  'a clause **where**  
*S\_Q* = *S\_M S* (*Q\_of\_state* (*Liminf\_state Sts*))

**interpretation** *sq*: *selection S\_Q*  
**unfolding** *S\_Q\_def* **using** *S\_M\_selects\_subseteq S\_M\_selects\_neg\_lits selection\_axioms*  
**by** *unfold\_locales auto*

**interpretation** *gr*: *ground\_resolution\_with\_selection S\_Q*  
**by** *unfold\_locales*

**interpretation** *sr*: *standard\_redundancy\_criterion\_reductive gr.ord\_Γ*  
**by** *unfold\_locales*

**interpretation** *sr*: *standard\_redundancy\_criterion\_counterex\_reducing gr.ord\_Γ*  
*ground\_resolution\_with\_selection.INTERP S\_Q*  
**by** *unfold\_locales*

The extension of ordered resolution mentioned in 4.10. We let it consist of all sound rules.

**definition** *ground\_sound\_Γ*:: 'a inference set **where**  
*ground\_sound\_Γ* = {*Infer CC D E* | *CC D E*. ( $\forall I. I \models_m CC \longrightarrow I \models D \longrightarrow I \models E$ )}

We prove that we indeed defined an extension.

**lemma** *gd\_ord\_Γ\_ngd\_ord\_Γ*: *gr.ord\_Γ*  $\subseteq$  *ground\_sound\_Γ*  
**unfolding** *ground\_sound\_Γ\_def* **using** *gr.ord\_Γ\_def gr.ord\_resolve\_sound* **by** *fastforce*

**lemma** *sound\_ground\_sound\_Γ*: *sound\_inference\_system ground\_sound\_Γ*  
**unfolding** *sound\_inference\_system\_def ground\_sound\_Γ\_def* **by** *auto*

**lemma** *sat\_preserving\_ground\_sound\_Γ*: *sat\_preserving\_inference\_system ground\_sound\_Γ*  
**using** *sound\_ground\_sound\_Γ sat\_preserving\_inference\_system.intro*  
*sound\_inference\_system.Γ\_sat\_preserving* **by** *blast*

**definition** *sr\_ext\_Ri* :: 'a clause set  $\Rightarrow$  'a inference set **where**  
*sr\_ext\_Ri N* = *sr.Ri N*  $\cup$  (*ground\_sound\_Γ* - *gr.ord\_Γ*)

**interpretation** *sr\_ext*:  
*sat\_preserving\_redundancy\_criterion ground\_sound\_Γ sr.Rf sr\_ext\_Ri*  
**unfolding** *sat\_preserving\_redundancy\_criterion\_def sr\_ext\_Ri\_def*  
**using** *sat\_preserving\_ground\_sound\_Γ redundancy\_criterion\_standard\_extension gd\_ord\_Γ\_ngd\_ord\_Γ*  
*sr.redundancy\_criterion\_axioms* **by** *auto*

**lemma** *strict\_subset\_subsumption\_redundant\_clause*:

**assumes**

*sub*: *D* ·  $\sigma \subset \#$  *C* **and**

*ground\_σ*: *is\_ground\_subst*  $\sigma$

**shows** *C*  $\in$  *sr.Rf* (*grounding\_of\_cls D*)

**proof** –

**from** *sub* **have**  $\forall I. I \models D \cdot \sigma \longrightarrow I \models C$

**unfolding** *true\_cls\_def* **by** *blast*

**moreover** **have** *C*  $>$  *D* ·  $\sigma$

**using** *sub* **by** (*simp add: subset\_imp\_less\_mset*)

**moreover** **have** *D* ·  $\sigma \in$  *grounding\_of\_cls D*

**using** *ground\_σ* **by** (*metis* (*mono\_tags*, *lifting*) *mem\_Collect\_eq substitution\_ops.grounding\_of\_cls\_def*)

**ultimately** **have** *set\_mset* { $\#D \cdot \sigma\#$ }  $\subseteq$  *grounding\_of\_cls D*

( $\forall I. I \models_m \{\#D \cdot \sigma\# \longrightarrow I \models C$ )

( $\forall D'. D' \in \# \{\#D \cdot \sigma\# \longrightarrow D' < C$ )

**by** *auto*

**then show** *?thesis*

**using** *sr.Rf\_def* **by** *blast*

**qed**

```

lemma strict_subset_subsumption_redundant_state:
  assumes
     $D \cdot \sigma \subset\# C$  and
    is_ground_subst  $\sigma$  and
     $D \in \text{clss\_of\_state } St$ 
  shows  $C \in \text{sr.Rf } (\text{grounding\_of\_state } St)$ 
  using assms
proof (induction  $St$ )
  case (fields  $N P Q$ )
  note  $sub = \text{this}(1)$  and  $gr = \text{this}(2)$  and  $d\_in = \text{this}(3)$ 

  have  $C \in \text{sr.Rf } (\text{grounding\_of\_cls } D)$ 
  by (rule strict_subset_subsumption_redundant_clause[OF  $sub\ gr$ ])
  then show ?case
    using  $d\_in$  unfolding clss_of_state_def grounding_of_cls_def
    by (metis (no_types) sr.Rf_mono sup_ge1 SUP_absorb contra_subsetD)
qed

```

```

lemma subst_cls_eq_grounding_of_cls_subset_eq:
  assumes  $D \cdot \sigma = C$ 
  shows  $\text{grounding\_of\_cls } C \subseteq \text{grounding\_of\_cls } D$ 
proof
  fix  $C\sigma'$ 
  assume  $C\sigma' \in \text{grounding\_of\_cls } C$ 
  then obtain  $\sigma'$  where
     $C\sigma': C \cdot \sigma' = C\sigma'$  is_ground_subst  $\sigma'$ 
    unfolding grounding_of_cls_def by auto
  then have  $C \cdot \sigma' = D \cdot \sigma \cdot \sigma' \wedge \text{is\_ground\_subst } (\sigma \odot \sigma')$ 
    using assms by auto
  then show  $C\sigma' \in \text{grounding\_of\_cls } D$ 
    unfolding grounding_of_cls_def using  $C\sigma'(1)$  by force
qed

```

The following corresponds the part of Lemma 4.10 that states we have a theorem proving process:

```

lemma resolution_prover_ground_derive:
   $St \rightsquigarrow St' \implies \text{sr.ext.derive } (\text{grounding\_of\_state } St) (\text{grounding\_of\_state } St')$ 
proof (induction rule: RP.induct)
  case (tautology_deletion  $A C N P Q$ )
  {
    fix  $C\sigma$ 
    assume  $C\sigma \in \text{grounding\_of\_cls } C$ 
    then obtain  $\sigma$  where
       $C\sigma = C \cdot \sigma$ 
      unfolding grounding_of_cls_def by auto
    then have  $\text{Neg } (A \cdot a \sigma) \in\# C\sigma \wedge \text{Pos } (A \cdot a \sigma) \in\# C\sigma$ 
      using tautology_deletion Neg_Melem_subst_atm_subst_cls Pos_Melem_subst_atm_subst_cls by auto
    then have  $C\sigma \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
      using sr.tautology_redundant by auto
  }
  then have  $\text{grounding\_of\_state } (N \cup \{C\}, P, Q) - \text{grounding\_of\_state } (N, P, Q)$ 
     $\subseteq \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
    unfolding clss_of_state_def grounding_of_cls_def by auto
  moreover have  $\text{grounding\_of\_state } (N, P, Q) - \text{grounding\_of\_state } (N \cup \{C\}, P, Q) = \{\}$ 
    unfolding clss_of_state_def grounding_of_cls_def by auto
  ultimately show ?case
    using sr.ext.derive.intros[of  $\text{grounding\_of\_state } (N, P, Q) \text{ grounding\_of\_state } (N \cup \{C\}, P, Q)$ ]
    by auto
next
  case (forward_subsumption  $D P Q C N$ )
  note  $D\_p = \text{this}$ 
  then obtain  $\sigma$  where
     $D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$ 

```

```

    by (auto simp: subsumes_def subset_mset_def)
  then have  $D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$ 
    by (simp add: subset_mset_def)
  then show ?case
proof
  assume  $D \cdot \sigma = C$ 
  then have  $\text{grounding\_of\_cls } C \subseteq \text{grounding\_of\_cls } D$ 
    using subst_cls_eq_grounding_of_cls_subset_eq by simp
  then have  $\text{grounding\_of\_state } (N \cup \{C\}, P, Q) = \text{grounding\_of\_state } (N, P, Q)$ 
    using D_p unfolding clss_of_state_def grounding_of_clss_def by auto
  then show ?case
    by (auto intro: sr_ext.derive.intros)
next
  assume a:  $D \cdot \sigma \subset\# C$ 
  have  $\text{grounding\_of\_cls } C \subseteq \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
proof
  fix  $C\mu$ 
  assume  $C\mu \in \text{grounding\_of\_cls } C$ 
  then obtain  $\mu$  where
     $\mu\text{-p}: C\mu = C \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto
  have  $D\sigma\mu C\mu: D \cdot \sigma \cdot \mu \subset\# C \cdot \mu$ 
    using a subst_subset_mono by auto
  then show  $C\mu \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
    using  $\mu\text{-p}$  strict_subset.subsumption_redundant_state[of  $D \sigma \odot \mu C \cdot \mu (N, P, Q)$ ] D_p
    unfolding clss_of_state_def by auto
qed
then show ?case
  unfolding clss_of_state_def grounding_of_clss_def by (force intro: sr_ext.derive.intros)
qed
next
case (backward_subsumption_P D N C P Q)

note D_p = this
then obtain  $\sigma$  where
   $D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$ 
  by (auto simp: strictly_subsumes_def subsumes_def subset_mset_def)
then show ?case
proof
  assume  $D \cdot \sigma = C$ 
  then have  $\text{grounding\_of\_cls } C \subseteq \text{grounding\_of\_cls } D$ 
    using subst_cls_eq_grounding_of_cls_subset_eq by simp
  then have  $\text{grounding\_of\_state } (N, P \cup \{C\}, Q) = \text{grounding\_of\_state } (N, P, Q)$ 
    using D_p unfolding clss_of_state_def grounding_of_clss_def by auto
  then show ?case
    by (auto intro: sr_ext.derive.intros)
next
  assume a:  $D \cdot \sigma \subset\# C$ 
  have  $\text{grounding\_of\_cls } C \subseteq \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
proof
  fix  $C\mu$ 
  assume  $C\mu \in \text{grounding\_of\_cls } C$ 
  then obtain  $\mu$  where
     $\mu\text{-p}: C\mu = C \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto
  have  $D\sigma\mu C\mu: D \cdot \sigma \cdot \mu \subset\# C \cdot \mu$ 
    using a subst_subset_mono by auto
  then show  $C\mu \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
    using  $\mu\text{-p}$  strict_subset.subsumption_redundant_state[of  $D \sigma \odot \mu C \cdot \mu (N, P, Q)$ ] D_p
    unfolding clss_of_state_def by auto
qed
then show ?case
  unfolding clss_of_state_def grounding_of_clss_def by (force intro: sr_ext.derive.intros)

```



```

qed
next
case (backward_subsumption_Q D N C P Q)
note D_p = this
then obtain  $\sigma$  where
   $D \cdot \sigma = C \vee D \cdot \sigma \subset\# C$ 
  by (auto simp: strictly_subsumes_def subsumes_def subset_mset_def)
then show ?case
proof
  assume  $D \cdot \sigma = C$ 
  then have grounding_of_cls  $C \subseteq \text{grounding\_of\_cls } D$ 
    using subst_cls_eq_grounding_of_cls_subset_eq by simp
  then have grounding_of_state  $(N, P, Q \cup \{C\}) = \text{grounding\_of\_state } (N, P, Q)$ 
    using D_p unfolding cls_of_state_def grounding_of_cls_def by auto
  then show ?case
    by (auto intro: sr_ext.derive.intros)
next
assume a:  $D \cdot \sigma \subset\# C$ 
have grounding_of_cls  $C \subseteq \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
proof
  fix  $C\mu$ 
  assume  $C\mu \in \text{grounding\_of\_cls } C$ 
  then obtain  $\mu$  where
     $\mu_p: C\mu = C \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto
  have  $D\sigma\mu C\mu: D \cdot \sigma \cdot \mu \subset\# C \cdot \mu$ 
    using a subst_subset_mono by auto
  then show  $C\mu \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P, Q))$ 
    using  $\mu_p$  strict_subset_subsumption_redundant_state[of D  $\sigma \odot \mu$   $C \cdot \mu$   $(N, P, Q)$ ] D_p
    unfolding cls_of_state_def by auto
qed
then show ?case
  unfolding cls_of_state_def grounding_of_cls_def by (force intro: sr_ext.derive.intros)
qed
next
case (forward_reduction D L' P Q L  $\sigma$  C N)
note DL'_p = this
{
  fix  $C\mu$ 
  assume  $C\mu \in \text{grounding\_of\_cls } C$ 
  then obtain  $\mu$  where
     $\mu_p: C\mu = C \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto

  define  $\gamma$  where
     $\gamma = \text{Infer } \{\#(C + \{\#L\#\}) \cdot \mu\# \} ((D + \{\#L'\#\}) \cdot \sigma \cdot \mu) (C \cdot \mu)$ 

  have  $(D + \{\#L'\#\}) \cdot \sigma \cdot \mu \in \text{grounding\_of\_state } (N \cup \{C + \{\#L\#\}\}, P, Q)$ 
    unfolding cls_of_state_def grounding_of_cls_def grounding_of_cls_def
    by (rule UN_I[of D +  $\{\#L'\#\}$ , use DL'_p(1) in simp,
      metis (mono_tags, lifting)  $\mu_p$  is_ground_comp_subst mem_Collect_eq subst_cls_comp_subst)
  moreover have  $(C + \{\#L\#\}) \cdot \mu \in \text{grounding\_of\_state } (N \cup \{C + \{\#L\#\}\}, P, Q)$ 
    using  $\mu_p$  unfolding cls_of_state_def grounding_of_cls_def grounding_of_cls_def by auto
  moreover have  $\forall I. I \models D \cdot \sigma \cdot \mu + \{\# - (L \cdot l \mu)\#\} \longrightarrow I \models C \cdot \mu + \{\#L \cdot l \mu\#\} \longrightarrow I \models D \cdot \sigma \cdot \mu + C$ 
    by auto
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models D \cdot \sigma \cdot \mu + C \cdot \mu$ 
    using DL'_p
    by (metis add_mset_add_single subst_cls_add_mset subst_cls_union subst_minus)
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    using DL'_p by (metis (no_types, lifting) subset_mset.le_iff_add subst_cls_union true_cls_union)
  then have  $\forall I. I \models m \{\#(D + \{\#L'\#\}) \cdot \sigma \cdot \mu\# \} \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    by (meson true_cls_mset_singleton)
}

```

```

ultimately have  $\gamma \in sr\_ext.inferences\_from (grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q))$ 
  unfolding  $sr\_ext.inferences\_from\_def$  unfolding  $ground\_sound\_I\_def$   $infer\_from\_def$   $\gamma\_def$  by auto
then have  $C \cdot \mu \in concls\_of (sr\_ext.inferences\_from (grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q)))$ 
  using  $image\_iff$  unfolding  $\gamma\_def$  by fastforce
then have  $C\mu \in concls\_of (sr\_ext.inferences\_from (grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q)))$ 
  using  $\mu\_p$  by auto
}
then have  $grounding\_of\_state (N \cup \{C\}, P, Q) - grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q)$ 
   $\subseteq concls\_of (sr\_ext.inferences\_from (grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q)))$ 
  unfolding  $grounding\_of\_clss\_def$   $clss\_of\_state\_def$  by auto
moreover
{
  fix  $CL\mu$ 
  assume  $CL\mu \in grounding\_of\_cls (C + \{\#L\#\})$ 
  then obtain  $\mu$  where
     $\mu\_def: CL\mu = (C + \{\#L\#\}) \cdot \mu \wedge is\_ground\_subst \mu$ 
    unfolding  $grounding\_of\_cls\_def$  by auto
  have  $C\mu\_CL\mu: C \cdot \mu \subsetneq (C + \{\#L\#\}) \cdot \mu$ 
    by auto
  then have  $(C + \{\#L\#\}) \cdot \mu \in sr.Rf (grounding\_of\_state (N \cup \{C\}, P, Q))$ 
    using  $sr.Rf\_def[of grounding\_of\_cls C]$ 
    using  $strict\_subset\_subsumption\_redundant\_state[of C \mu (C + \{\#L\#\}) \cdot \mu (N \cup \{C\}, P, Q)] \mu\_def$ 
    unfolding  $clss\_of\_state\_def$  by force
  then have  $CL\mu \in sr.Rf (grounding\_of\_state (N \cup \{C\}, P, Q))$ 
    using  $\mu\_def$  by auto
}
then have  $grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q) - grounding\_of\_state (N \cup \{C\}, P, Q)$ 
   $\subseteq sr.Rf (grounding\_of\_state (N \cup \{C\}, P, Q))$ 
  unfolding  $clss\_of\_state\_def$   $grounding\_of\_clss\_def$  by auto
ultimately show ?case
  using  $sr\_ext.derive.intros[of grounding\_of\_state (N \cup \{C\}, P, Q)$ 
     $grounding\_of\_state (N \cup \{C + \{\#L\#\}\}, P, Q)]$ 
  by auto
next
case  $(backward\_reduction\_P D L' N L \sigma C P Q)$ 
note  $DL'_p = this$ 
{
  fix  $C\mu$ 
  assume  $C\mu \in grounding\_of\_cls C$ 
  then obtain  $\mu$  where
     $\mu\_p: C\mu = C \cdot \mu \wedge is\_ground\_subst \mu$ 
    unfolding  $grounding\_of\_cls\_def$  by auto

  define  $\gamma$  where
     $\gamma = Infer \{\#(C + \{\#L\#\}) \cdot \mu\# \} ((D + \{\#L'\#\}) \cdot \sigma \cdot \mu) (C \cdot \mu)$ 

  have  $(D + \{\#L'\#\}) \cdot \sigma \cdot \mu \in grounding\_of\_state (N, P \cup \{C + \{\#L\#\}\}, Q)$ 
    unfolding  $clss\_of\_state\_def$   $grounding\_of\_clss\_def$   $grounding\_of\_cls\_def$ 
    by  $(rule UN\_I[of D + \{\#L'\#\}], use DL'_p(1) in simp,$ 
       $metis (mono\_tags, lifting) \mu\_p is\_ground\_comp\_subst mem\_Collect\_eq subst\_cls\_comp\_subst)$ 
  moreover have  $(C + \{\#L\#\}) \cdot \mu \in grounding\_of\_state (N, P \cup \{C + \{\#L\#\}\}, Q)$ 
    using  $\mu\_p$  unfolding  $clss\_of\_state\_def$   $grounding\_of\_clss\_def$   $grounding\_of\_cls\_def$  by auto
  moreover have  $\forall I. I \models D \cdot \sigma \cdot \mu + \{\#-(L \cdot l \mu)\#\} \longrightarrow I \models C \cdot \mu + \{\#L \cdot l \mu\#\} \longrightarrow I \models D \cdot \sigma \cdot \mu + C$ 
    by auto
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models D \cdot \sigma \cdot \mu + C \cdot \mu$ 
    using  $DL'_p$ 
    by  $(metis add\_mset\_add\_single subst\_cls\_add\_mset subst\_cls\_union subst\_minus)$ 
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    using  $DL'_p$  by  $(metis (no\_types, lifting) subset\_mset.le\_iff\_add subst\_cls\_union true\_cls\_union)$ 
  then have  $\forall I. I \models m \{\#(D + \{\#L'\#\}) \cdot \sigma \cdot \mu\# \} \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    by  $(meson true\_cls\_mset\_singleton)$ 
  ultimately have  $\gamma \in sr\_ext.inferences\_from (grounding\_of\_state (N, P \cup \{C + \{\#L\#\}\}, Q))$ 

```

```

    unfolding sr_ext.inferences_from_def unfolding ground_sound_Γ_def infer_from_def γ_def by simp
  then have  $C \cdot \mu \in \text{concls\_of } (sr\_ext.inferences\_from (grounding\_of\_state (N, P \cup \{C + \{\#L\#\}, Q)))$ 
    using image_iff unfolding γ_def by fastforce
  then have  $C\mu \in \text{concls\_of } (sr\_ext.inferences\_from (grounding\_of\_state (N, P \cup \{C + \{\#L\#\}, Q)))$ 
    using μ_p by auto
}
then have  $\text{grounding\_of\_state } (N, P \cup \{C\}, Q) - \text{grounding\_of\_state } (N, P \cup \{C + \{\#L\#\}, Q)$ 
   $\subseteq \text{concls\_of } (sr\_ext.inferences\_from (grounding\_of\_state (N, P \cup \{C + \{\#L\#\}, Q)))$ 
  unfolding grounding_of_cls_def cls_of_state_def by auto
moreover
{
  fix CLμ
  assume  $CL\mu \in \text{grounding\_of\_cls } (C + \{\#L\#\})$ 
  then obtain μ where
    μ_def:  $CL\mu = (C + \{\#L\#\}) \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto
  have  $C\mu \cdot CL\mu: C \cdot \mu \subseteq \# (C + \{\#L\#\}) \cdot \mu$ 
    by auto
  then have  $(C + \{\#L\#\}) \cdot \mu \in sr.Rf (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
    using sr.Rf_def[of grounding_of_cls C]
    using strict_subset_subsumption_redundant_state[of C μ  $(C + \{\#L\#\}) \cdot \mu$   $(N, P \cup \{C\}, Q)$ ] μ_def
    unfolding cls_of_state_def by force
  then have  $CL\mu \in sr.Rf (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
    using μ_def by auto
}
then have  $\text{grounding\_of\_state } (N, P \cup \{C + \{\#L\#\}, Q) - \text{grounding\_of\_state } (N, P \cup \{C\}, Q)$ 
   $\subseteq sr.Rf (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
  unfolding cls_of_state_def grounding_of_cls_def by auto
ultimately show ?case
  using sr_ext.derive.intros[of grounding_of_state  $(N, P \cup \{C\}, Q)$ 
    grounding_of_state  $(N, P \cup \{C + \{\#L\#\}, Q)$ ]
  by auto
next
case (backward_reduction_Q D L' N L σ C P Q)
note DL'_p = this
{
  fix Cμ
  assume  $C\mu \in \text{grounding\_of\_cls } C$ 
  then obtain μ where
    μ_p:  $C\mu = C \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
    unfolding grounding_of_cls_def by auto

  define γ where
     $\gamma = \text{Infer } \{\#(C + \{\#L\#\}) \cdot \mu\# \} ((D + \{\#L'\#\}) \cdot \sigma \cdot \mu) (C \cdot \mu)$ 

  have  $(D + \{\#L'\#\}) \cdot \sigma \cdot \mu \in \text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})$ 
    unfolding cls_of_state_def grounding_of_cls_def
    by (rule UN_I[of D +  $\{\#L'\#\}$ , use DL'_p(1) in simp,
      metis (mono_tags, lifting) μ_p is_ground_comp_subst mem_Collect_eq subst_cls_comp_subst)
  moreover have  $(C + \{\#L\#\}) \cdot \mu \in \text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})$ 
    using μ_p unfolding cls_of_state_def grounding_of_cls_def grounding_of_cls_def by auto
  moreover have  $\forall I. I \models D \cdot \sigma \cdot \mu + \{\#- (L \cdot l \mu)\#\} \longrightarrow I \models C \cdot \mu + \{\#L \cdot l \mu\#\} \longrightarrow I \models D \cdot \sigma \cdot \mu + C$ 
    by auto
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models D \cdot \sigma \cdot \mu + C \cdot \mu$ 
    using DL'_p
    by (metis add_mset_add_single subst_cls_add_mset subst_cls_union subst_minus)
  then have  $\forall I. I \models (D + \{\#L'\#\}) \cdot \sigma \cdot \mu \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    using DL'_p by (metis (no_types, lifting) subset_mset.le_iff_add subst_cls_union true_cls_union)
  then have  $\forall I. I \models m \{\#(D + \{\#L'\#\}) \cdot \sigma \cdot \mu\# \} \longrightarrow I \models (C + \{\#L\#\}) \cdot \mu \longrightarrow I \models C \cdot \mu$ 
    by (meson true_cls_mset_singleton)
  ultimately have  $\gamma \in sr\_ext.inferences\_from (\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\}))$ 
    unfolding sr_ext.inferences_from_def unfolding ground_sound_Γ_def infer_from_def γ_def by simp
}

```

```

then have  $C \cdot \mu \in \text{concls\_of } (\text{sr\_ext.inferences\_from } (\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})))$ 
using image_iff unfolding  $\gamma\_def$  by fastforce
then have  $C\mu \in \text{concls\_of } (\text{sr\_ext.inferences\_from } (\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})))$ 
using  $\mu\_p$  by auto
}
then have  $\text{grounding\_of\_state } (N, P \cup \{C\}, Q) - \text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})$ 
 $\subseteq \text{concls\_of } (\text{sr\_ext.inferences\_from } (\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})))$ 
unfolding grounding\_of\_clss\_def clss\_of\_state\_def by auto
moreover
{
  fix  $CL\mu$ 
assume  $CL\mu \in \text{grounding\_of\_cls } (C + \{\#L\#\})$ 
then obtain  $\mu$  where
   $\mu\_def: CL\mu = (C + \{\#L\#\}) \cdot \mu \wedge \text{is\_ground\_subst } \mu$ 
unfolding grounding\_of\_cls\_def by auto
have  $C\mu\_CL\mu: C \cdot \mu \subseteq \# (C + \{\#L\#\}) \cdot \mu$ 
by auto
then have  $(C + \{\#L\#\}) \cdot \mu \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
using sr.Rf\_def[of grounding\_of\_cls C]
using strict\_subset\_subsumption\_redundant\_state[of C  $\mu$   $(C + \{\#L\#\}) \cdot \mu$   $(N, P \cup \{C\}, Q)$   $\mu\_def$ 
unfolding clss\_of\_state\_def by force
then have  $CL\mu \in \text{sr.Rf } (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
using  $\mu\_def$  by auto
}
then have  $\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\}) - \text{grounding\_of\_state } (N, P \cup \{C\}, Q)$ 
 $\subseteq \text{sr.Rf } (\text{grounding\_of\_state } (N, P \cup \{C\}, Q))$ 
unfolding clss\_of\_state\_def grounding\_of\_clss\_def by auto
ultimately show ?case
using sr\_ext.derive.intros[of grounding\_of\_state  $(N, P \cup \{C\}, Q)$   $\text{grounding\_of\_state } (N, P, Q \cup \{C + \{\#L\#\}\})$ ]
by auto
next
case (clause\_processing N C P Q)
then show ?case
unfolding clss\_of\_state\_def using sr\_ext.derive.intros by auto
next
case (inference\_computation N Q C P)
{
  fix  $E\mu$ 
assume  $E\mu \in \text{grounding\_of\_clss } N$ 
then obtain  $\mu$   $E$  where
   $E\_ \mu\_p: E\mu = E \cdot \mu \wedge E \in N \wedge \text{is\_ground\_subst } \mu$ 
unfolding grounding\_of\_clss\_def grounding\_of\_cls\_def by auto
then have  $E\_concl: E \in \text{concls\_of } (\text{ord\_FO\_resolution.inferences\_between } Q C)$ 
using inference\_computation by auto
then obtain  $\gamma$  where
   $\gamma\_p: \gamma \in \text{ord\_FO}\Gamma S \wedge \text{infer\_from } (Q \cup \{C\}) \gamma \wedge C \in \# \text{prems\_of } \gamma \wedge \text{concl\_of } \gamma = E$ 
unfolding ord\_FO\_resolution.inferences\_between\_def by auto
then obtain  $CC$   $CAs$   $D$   $AAs$   $As$   $\sigma$  where
   $\gamma\_p2: \gamma = \text{Infer } CC D E \wedge \text{ord\_resolve\_rename } S CAs D AAs As \sigma E \wedge \text{mset } CAs = CC$ 
unfolding ord\_FO\Gamma\_def by auto
define  $\varrho$  where
 $\varrho = \text{hd } (\text{renamings\_apart } (D \# CAs))$ 
define  $\varrho s$  where
 $\varrho s = \text{tl } (\text{renamings\_apart } (D \# CAs))$ 
define  $\gamma\_ground$  where
 $\gamma\_ground = \text{Infer } (\text{mset } (CAs \cdot\!\!\cdot\!\!\cdot \varrho s) \cdot \text{cm } \sigma \cdot \text{cm } \mu) (D \cdot \varrho \cdot \sigma \cdot \mu) (E \cdot \mu)$ 
have  $\forall I. I \models_m \text{mset } (CAs \cdot\!\!\cdot\!\!\cdot \varrho s) \cdot \text{cm } \sigma \cdot \text{cm } \mu \longrightarrow I \models D \cdot \varrho \cdot \sigma \cdot \mu \longrightarrow I \models E \cdot \mu$ 
using ord\_resolve\_rename\_ground\_inst\_sound[of - - - - -  $\mu$   $\varrho\_def$   $\varrho s\_def$   $E\_ \mu\_p$   $\gamma\_p2$ ]
by auto
then have  $\gamma\_ground \in \{\text{Infer } cc d e \mid cc d e. \forall I. I \models_m cc \longrightarrow I \models d \longrightarrow I \models e\}$ 
unfolding  $\gamma\_ground\_def$  by auto
moreover have  $\text{set\_mset } (\text{prems\_of } \gamma\_ground) \subseteq \text{grounding\_of\_state } (\{ \}, P \cup \{C\}, Q)$ 

```

```

proof –
  have  $D = C \vee D \in Q$ 
    unfolding  $\gamma\_ground\_def$  using  $E\_mu\_p$   $\gamma\_p2$   $\gamma\_p$  unfolding  $infer\_from\_def$ 
    unfolding  $clss\_of\_state\_def$   $grounding\_of\_clss\_def$ 
    unfolding  $grounding\_of\_cls\_def$ 
    by  $simp$ 
  then have  $D \cdot \varrho \cdot \sigma \cdot \mu \in grounding\_of\_cls\ C \vee (\exists x \in Q. D \cdot \varrho \cdot \sigma \cdot \mu \in grounding\_of\_cls\ x)$ 
    using  $E\_mu\_p$ 
    unfolding  $grounding\_of\_cls\_def$ 
    by ( $metis$  ( $mono\_tags$ ,  $lifting$ )  $is\_ground\_comp\_subst$   $mem\_Collect\_eq$   $subst\_cls\_comp\_subst$ )
  then have  $(D \cdot \varrho \cdot \sigma \cdot \mu \in grounding\_of\_cls\ C \vee$ 
     $(\exists x \in P. D \cdot \varrho \cdot \sigma \cdot \mu \in grounding\_of\_cls\ x) \vee$ 
     $(\exists x \in Q. D \cdot \varrho \cdot \sigma \cdot \mu \in grounding\_of\_cls\ x))$ 
    by  $metis$ 
  moreover have  $\forall i < length\ (CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu). ((CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu) ! i) \in$ 
     $\{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\} \cup$ 
     $((\bigcup C \in P. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}) \cup (\bigcup C \in Q. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}))$ 
  proof ( $rule$ ,  $rule$ )
    fix  $i$ 
    assume  $i < length\ (CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu)$ 
    then have  $a: i < length\ CAs \wedge i < length\ \varrho s$ 
      by  $simp$ 
    moreover from  $a$  have  $CAs ! i \in \{C\} \cup Q$ 
      using  $\gamma\_p2$   $\gamma\_p$  unfolding  $infer\_from\_def$ 
      by ( $metis$  ( $no\_types$ ,  $lifting$ )  $Un\_subset\_iff$   $inference.sel(1)$   $set\_mset\_union$ 
         $sup\_commute$   $nth\_mem\_mset$   $subsetCE$ )
    ultimately have  $(CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu) ! i \in$ 
       $\{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\} \vee$ 
       $((CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu) ! i \in (\bigcup C \in P. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}) \vee$ 
       $(CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu) ! i \in (\bigcup C \in Q. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}))$ 
      unfolding  $\gamma\_ground\_def$  using  $E\_mu\_p$   $\gamma\_p2$   $\gamma\_p$  unfolding  $infer\_from\_def$ 
      unfolding  $clss\_of\_state\_def$   $grounding\_of\_clss\_def$ 
      unfolding  $grounding\_of\_cls\_def$ 
      apply –
      apply ( $cases\ CAs ! i = C$ )
    subgoal
      apply ( $rule\ disjI1$ )
      apply ( $rule\ Set.CollectI$ )
      apply ( $rule\_tac\ x=(\varrho s ! i) \odot \sigma \odot \mu$  in  $exI$ )
      using  $\varrho s\_def$  using  $renames\_apart$  apply ( $auto;fail$ )
      done
    subgoal
      apply ( $rule\ disjI2$ )
      apply ( $rule\ disjI2$ )
      apply ( $rule\_tac\ a=CAs ! i$  in  $UN\_I$ )
    subgoal
      apply  $blast$ 
      done
    subgoal
      apply ( $rule\ Set.CollectI$ )
      apply ( $rule\_tac\ x=(\varrho s ! i) \odot \sigma \odot \mu$  in  $exI$ )
      using  $\varrho s\_def$  using  $renames\_apart$  apply ( $auto;fail$ )
      done
    done
    done
  then show  $(CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu) ! i \in \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\} \cup$ 
     $((\bigcup C \in P. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}) \cup (\bigcup C \in Q. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}))$ 
    by  $blast$ 
qed
then have  $\forall x \in \# mset\ (CAs \cdot cl\ \varrho s \cdot cl\ \sigma \cdot cl\ \mu). x \in \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\} \cup$ 
     $((\bigcup C \in P. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}) \cup (\bigcup C \in Q. \{C \cdot \sigma \mid \sigma. is\_ground\_subst\ \sigma\}))$ 
    by ( $metis$  ( $lifting$ )  $in\_set\_conv\_nth$   $set\_mset\_mset$ )
then have  $set\_mset\ (mset\ (CAs \cdot cl\ \varrho s) \cdot cm\ \sigma \cdot cm\ \mu) \subseteq$ 

```

```

    grounding_of_cls C ∪ grounding_of_cls P ∪ grounding_of_cls Q
    unfolding grounding_of_cls_def grounding_of_cls_def
    using mset_subst_cls_list_subst_cls_mset by auto
    ultimately show ?thesis
    unfolding γ_ground_def cls_of_state_def grounding_of_cls_def by auto
qed
ultimately have E · μ ∈ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
    unfolding sr_ext.inferences_from_def inference_system.inferences_from_def ground_sound_Γ_def infer_from_def
    using γ_ground_def by (metis (no_types, lifting) imageI inference.sel(3) mem_Collect_eq)
then have Eμ ∈ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
    using E_μ_p by auto
}
then have grounding_of_state (N, P, Q ∪ {C}) = grounding_of_state ({}, P ∪ {C}, Q)
    ⊆ concls_of (sr_ext.inferences_from (grounding_of_state ({}, P ∪ {C}, Q)))
    unfolding cls_of_state_def grounding_of_cls_def by auto
moreover have grounding_of_state ({}, P ∪ {C}, Q) = grounding_of_state (N, P, Q ∪ {C}) = {}
    unfolding cls_of_state_def grounding_of_cls_def by auto
ultimately show ?case
    using sr_ext.derive.intros[of (grounding_of_state (N, P, Q ∪ {C}))
    (grounding_of_state ({}, P ∪ {C}, Q))] by auto
qed

```

A useful consequence:

**lemma** *RP\_model*:  $St \rightsquigarrow St' \implies I \models_s \text{grounding\_of\_state } St' \longleftrightarrow I \models_s \text{grounding\_of\_state } St$

**proof** (*drule resolution\_prover\_ground\_derive, erule sr\_ext.derive.cases, hypsubst*)

let

?gSt = grounding\_of\_state St and

?gSt' = grounding\_of\_state St'

assume

deduct: ?gSt' - ?gSt ⊆ concls\_of (sr\_ext.inferences\_from ?gSt) (is - ⊆ ?concls) and

delete: ?gSt - ?gSt' ⊆ sr.Rf ?gSt'

show  $I \models_s ?gSt' \longleftrightarrow I \models_s ?gSt$

**proof**

assume bef:  $I \models_s ?gSt$

then have  $I \models_s ?concls$

unfolding ground\_sound\_Γ\_def inference\_system.inferences\_from\_def true\_cls\_def true\_cls\_mset\_def  
by (auto simp add: image\_def infer\_from\_def dest!: spec[of - I])

then have diff:  $I \models_s ?gSt' - ?gSt$

using deduct by (blast intro: true\_cls\_mono)

then show  $I \models_s ?gSt'$

using bef unfolding true\_cls\_def by blast

next

assume aft:  $I \models_s ?gSt'$

have  $I \models_s ?gSt' \cup sr.Rf ?gSt'$

by (rule sr.Rf\_model) (metis aft sr.Rf\_mono[OF Un\_upper1] Diff\_eq\_empty\_iff Diff\_subset  
Un\_Diff true\_cls\_mono true\_cls\_union)

then have  $I \models_s sr.Rf ?gSt'$

using true\_cls\_union by blast

then have diff:  $I \models_s ?gSt - ?gSt'$

using delete by (blast intro: true\_cls\_mono)

then show  $I \models_s ?gSt$

using aft unfolding true\_cls\_def by blast

qed

qed

Another formulation of the part of Lemma 4.10 that states we have a theorem proving process:

**lemma** *resolution\_prover\_ground\_derivation*:

$\text{chain } (op \rightsquigarrow) \text{ } Sts \implies \text{chain } sr\_ext.derive \text{ } (lmap \text{ grounding\_of\_state } Sts)$

using resolution\_prover\_ground\_derive by (simp add: chain\_lmap[of op  $\rightsquigarrow$ ])

The following is used prove to Lemma 4.11:

**lemma** *in\_Sup\_llist\_in\_nth*:  $C \in \text{Sup\_llist } Ns \implies \exists j. \text{enat } j < \text{llength } Ns \wedge C \in \text{lnth } Ns \ j$   
**unfolding** *Sup\_llist\_def* **by** *auto*

**lemma** *Sup\_llist\_grounding\_of\_state\_ground*:  
**assumes**  $C \in \text{Sup\_llist } (\text{lmap } \text{grounding\_of\_state } Sts)$   
**shows** *is\_ground\_cls*  $C$

**proof** –  
**have**  $\exists j. \text{enat } j < \text{llength } (\text{lmap } \text{grounding\_of\_state } Sts) \wedge C \in \text{lnth } (\text{lmap } \text{grounding\_of\_state } Sts) \ j$   
**using** *assms in\_Sup\_llist\_in\_nth* **by** *metis*  
**then obtain**  $j$  **where**  
 $\text{enat } j < \text{llength } (\text{lmap } \text{grounding\_of\_state } Sts)$   
 $C \in \text{lnth } (\text{lmap } \text{grounding\_of\_state } Sts) \ j$   
**by** *blast*  
**then show** *?thesis*  
**unfolding** *grounding\_of\_cls\_def* *grounding\_of\_cls\_def* **by** *auto*  
**qed**

**lemma** *Liminf\_grounding\_of\_state\_ground*:  
 $C \in \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_state } Sts) \implies \text{is\_ground\_cls } C$   
**using** *Liminf\_llist\_subset\_Sup\_llist* [of *lmap* *grounding\_of\_state* *Sts*]  
*Sup\_llist\_grounding\_of\_state\_ground*  
**by** *blast*

**lemma** *in\_Sup\_llist\_in\_Sup\_state*:  
**assumes**  $C \in \text{Sup\_llist } (\text{lmap } \text{grounding\_of\_state } Sts)$   
**shows**  $\exists D \ \sigma. D \in \text{cls\_of\_state } (\text{Sup\_state } Sts) \wedge D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$

**proof** –  
**from** *assms* **obtain**  $i$  **where**  
 $i\_p: \text{enat } i < \text{llength } Sts \wedge C \in \text{lnth } (\text{lmap } \text{grounding\_of\_state } Sts) \ i$   
**using** *in\_Sup\_llist\_in\_nth* **by** *fastforce*  
**then obtain**  $D \ \sigma$  **where**  
 $D \in \text{cls\_of\_state } (\text{lnth } Sts \ i) \wedge D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$   
**using** *assms* **unfolding** *grounding\_of\_cls\_def* *grounding\_of\_cls\_def* **by** *fastforce*  
**then have**  $D \in \text{cls\_of\_state } (\text{Sup\_state } Sts) \wedge D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$   
**using**  $i\_p$  **unfolding** *Sup\_state\_def* *cls\_of\_state\_def*  
**by** (*metis* (*no\\_types*, *lifting*) *UnCI* *UnE* *contra\_subsetD* *N\_of\_state\_simps* *P\_of\_state\_simps* *Q\_of\_state\_simps* *llength\_lmap* *lnth\_lmap* *lnth\_subset\_Sup\_llist*)  
**then show** *?thesis*  
**by** *auto*  
**qed**

**lemma**  
 $N\_of\_state\_Liminf: N\_of\_state (\text{Liminf\_state } Sts) = \text{Liminf\_llist } (\text{lmap } N\_of\_state \ Sts)$  **and**  
 $P\_of\_state\_Liminf: P\_of\_state (\text{Liminf\_state } Sts) = \text{Liminf\_llist } (\text{lmap } P\_of\_state \ Sts)$   
**unfolding** *Liminf\_state\_def* **by** *auto*

**lemma** *eventually\_removed\_from\_N*:  
**assumes**  
 $d\_in: D \in N\_of\_state (\text{lnth } Sts \ i)$  **and**  
 $fair: fair\_state\_seq \ Sts$  **and**  
 $i\_Sts: \text{enat } i < \text{llength } Sts$   
**shows**  $\exists l. D \in N\_of\_state (\text{lnth } Sts \ l) \wedge D \notin N\_of\_state (\text{lnth } Sts \ (\text{Suc } l)) \wedge i \leq l \wedge \text{enat } (\text{Suc } l) < \text{llength } Sts$

**proof** (*rule ccontr*)  
**assume**  $a: \neg ?thesis$   
**have**  $i \leq l \implies \text{enat } l < \text{llength } Sts \implies D \in N\_of\_state (\text{lnth } Sts \ l)$  **for**  $l$   
**using**  $d\_in$  **by** (*induction*  $l$ , *blast*, *metis*  $a$  *Suc\_ile\_eq* *le\_SucE* *less\_imp\_le*)  
**then have**  $D \in \text{Liminf\_llist } (\text{lmap } N\_of\_state \ Sts)$   
**unfolding** *Liminf\_llist\_def* **using**  $i\_Sts$  **by** *auto*  
**then show** *False*  
**using**  $fair$  **unfolding** *fair\_state\_seq\_def* **by** (*simp* *add: N\_of\_state\_Liminf*)  
**qed**

**lemma** *eventually\_removed\_from\_P*:

```

assumes
  d_in:  $D \in P\_of\_state\ (lnth\ Sts\ i)$  and
  fair: fair_state_seq Sts and
  i_Sts: enat i < llength Sts
shows  $\exists l. D \in P\_of\_state\ (lnth\ Sts\ l) \wedge D \notin P\_of\_state\ (lnth\ Sts\ (Suc\ l)) \wedge i \leq l \wedge enat\ (Suc\ l) < llength\ Sts$ 
proof (rule ccontr)
assume a:  $\neg ?thesis$ 
have  $i \leq l \implies enat\ l < llength\ Sts \implies D \in P\_of\_state\ (lnth\ Sts\ l)$  for l
  using d_in by (induction l, blast, metis a Suc_ile_eq le_SucE less_imp_le)
then have  $D \in Liminf\_llist\ (lmap\ P\_of\_state\ Sts)$ 
  unfolding Liminf_llist_def using i_Sts by auto
then show False
  using fair unfolding fair_state_seq_def by (simp add: P_of_state_Liminf)
qed

```

**lemma** *instance\_if\_subsumed\_and\_in\_limit*:

```

assumes
  ns:  $Ns = lmap\ grounding\_of\_state\ Sts$  and
  c:  $C \in Liminf\_llist\ Ns - sr.Rf\ (Liminf\_llist\ Ns)$  and

  d:  $D \in N\_of\_state\ (lnth\ Sts\ i) \cup P\_of\_state\ (lnth\ Sts\ i) \cup Q\_of\_state\ (lnth\ Sts\ i)$ 
  enat i < llength Sts subsumes D C
shows  $\exists \sigma. D \cdot \sigma = C \wedge is\_ground\_subst\ \sigma$ 
proof -
  let ?Ps =  $\lambda i. P\_of\_state\ (lnth\ Sts\ i)$ 
  let ?Qs =  $\lambda i. Q\_of\_state\ (lnth\ Sts\ i)$ 

  have ground_C: is_ground_cls C
    using c using Liminf_grounding_of_state_ground ns by auto

  have derivns: chain sr_ext.derive Ns
    using resolution_prover_ground_derivation deriv ns by auto

  have  $\exists \sigma. D \cdot \sigma = C$ 
proof (rule ccontr)
  assume  $\nexists \sigma. D \cdot \sigma = C$ 
  moreover from d(β) obtain τ_proto where
     $D \cdot \tau\_proto \subseteq\# C$  unfolding subsumes_def
    by blast
  then obtain τ where
     $\tau\_p: D \cdot \tau \subseteq\# C \wedge is\_ground\_subst\ \tau$ 
    using ground_C by (metis is_ground_cls_mono make_ground_subst subset_mset.order_refl)
  ultimately have subsub:  $D \cdot \tau \subseteq\# C$ 
    using subset_mset.le_imp_less_or_eq by auto
  moreover have is_ground_subst τ
    using τ_p by auto
  moreover have  $D \in cls\_of\_state\ (lnth\ Sts\ i)$ 
    using d unfolding cls_of_state_def by auto
  ultimately have  $C \in sr.Rf\ (grounding\_of\_state\ (lnth\ Sts\ i))$ 
    using strict_subset_subsumption_redundant_state[of D τ C lnth Sts i] by auto
  then have  $C \in sr.Rf\ (Sup\_llist\ Ns)$ 
    using d ns by (metis contra_subsetD llength_lmap lnth_lmap lnth_subset_Sup_llist sr.Rf_mono)
  then have  $C \in sr.Rf\ (Liminf\_llist\ Ns)$ 
    unfolding ns using local.sr_ext.Rf_Sup_subset_Rf_Liminf derivns ns by auto
  then show False
    using c by auto
qed
then obtain σ where
   $D \cdot \sigma = C \wedge is\_ground\_subst\ \sigma$ 
  using ground_C by (metis make_ground_subst)
then show ?thesis
  by auto

```



qed

**lemma** *from\_Q\_to\_Q\_inf*:

**assumes**

*fair*: *fair\_state\_seq* *Sts* **and**

*ns*: *Ns* = *lmap* *grounding\_of\_state* *Sts* **and**

*c*: *C* ∈ *Liminf\_llist* *Ns* − *sr.Rf* (*Liminf\_llist* *Ns*) **and**

*d*: *D* ∈ *Q\_of\_state* (*lnth* *Sts* *i*) *enat* *i* < *llength* *Sts* *subsumes* *D* *C* **and**

*d.least*:  $\forall E. E \in (\text{clss\_of\_state } (\text{Sup\_state } \text{Sts})) \wedge \text{subsumes } E \ C \}. \neg \text{strictly\_subsumes } E \ D$

**shows** *D* ∈ *Q\_of\_state* (*Liminf\_state* *Sts*)

**proof** −

**let** *?Ps* =  $\lambda i. P\_of\_state \ (lnth \ Sts \ i)$

**let** *?Qs* =  $\lambda i. Q\_of\_state \ (lnth \ Sts \ i)$

**have** *ground\_C*: *is\_ground\_cls* *C*

**using** *c* **using** *Liminf\_grounding\_of\_state\_ground* *ns* **by** *auto*

**have** *derivns*: *chain* *sr\_ext.derive* *Ns*

**using** *resolution\_prover\_ground\_derivation* *deriv* *ns* **by** *auto*

**have**  $\exists \sigma. D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$

**using** *instance\_if\_subsumed\_and\_in\_limit* *ns* *c* *d* **by** *blast*

**then obtain**  $\sigma$  **where**

$\sigma: D \cdot \sigma = C \text{ is\_ground\_subst } \sigma$

**by** *auto*

**from** *deriv* **have** *four\_ten*: *chain* *sr\_ext.derive* *Ns*

**using** *resolution\_prover\_ground\_derivation* *ns* **by** *auto*

**have** *in\_Sts.in\_Sts\_Suc*:

$\forall l \geq i. \text{enat } (\text{Suc } l) < \text{llength } \text{Sts} \longrightarrow D \in Q\_of\_state \ (lnth \ \text{Sts} \ l) \longrightarrow D \in Q\_of\_state \ (lnth \ \text{Sts} \ (\text{Suc } l))$

**proof** (*rule*, *rule*, *rule*, *rule*)

**fix** *l*

**assume**

*len*: *i* ≤ *l* **and**

*llen*: *enat* (*Suc* *l*) < *llength* *Sts* **and**

*d.in\_q*: *D* ∈ *Q\_of\_state* (*lnth* *Sts* *l*)

**have** *lnth* *Sts* *l*  $\rightsquigarrow$  *lnth* *Sts* (*Suc* *l*)

**using** *llen* *deriv* *chain\_lnth\_rel* **by** *blast*

**then show** *D* ∈ *Q\_of\_state* (*lnth* *Sts* (*Suc* *l*))

**proof** (*cases* *rule*: *RP.cases*)

**case** (*backward\_subsumption\_Q* *D' N D.removed* *P Q*)

**moreover**

{

**assume** *D.removed* = *D*

**then obtain** *D.subsumes* **where**

*D.subsumes\_p*: *D.subsumes* ∈ *N* ∧ *strictly\_subsumes* *D.subsumes* *D*

**using** *backward\_subsumption\_Q* **by** *auto*

**moreover from** *D.subsumes\_p* **have** *subsumes* *D.subsumes* *C*

**using** *d* *subsumes\_trans* **unfolding** *strictly\_subsumes\_def* **by** *blast*

**moreover from** *backward\_subsumption\_Q* **have** *D.subsumes* ∈ *clss\_of\_state* (*Sup\_state* *Sts*)

**using** *D.subsumes\_p* *llen*

**by** (*metis* (*no\_types*) *UnI1* *clss\_of\_state\_def* *N\_of\_state.simps* *llength\_lmap* *lnth\_lmap*

*lnth\_subset\_Sup\_llist* *rev\_subsetD* *Sup\_state\_def*)

**ultimately have** *False*

**using** *d.least* **unfolding** *subsumes\_def* **by** *auto*

}

**ultimately show** *?thesis*

**using** *d.in\_q* **by** *auto*

**next**

**case** (*backward\_reduction\_Q* *E L' N L σ D' P Q*)

{

```

    assume  $D' + \{\#L\# \} = D$ 
    then have  $D'_p$ : strictly_subsumes  $D' D \wedge D' \in ?Ps$  (Suc l)
      using subset_strictly_subsumes[of  $D' D$ ] backward_reduction_Q by auto
    then have subc: subsumes  $D' C$ 
      using  $d(3)$  subsumes_trans unfolding strictly_subsumes_def by auto
    from  $D'_p$  have  $D' \in \text{clss\_of\_state}$  (Sup\_state Sts)
      using llen by (metis (no\_types) UnI1 clss\_of\_state_def P\_of\_state_simps llen\_lmap
        lnth\_lmap lnth\_subset\_Sup\_llist subsetCE sup\_ge2 Sup\_state_def)
    then have False
      using d\_least  $D'_p$  subc by auto
  }
  then show ?thesis
    using backward_reduction_Q d\_in\_q by auto
qed (use d\_in\_q in auto)
qed
have  $D\_in\_Sts$ :  $D \in Q\_of\_state$  (lnth Sts l) and  $D\_in\_Sts\_Suc$ :  $D \in Q\_of\_state$  (lnth Sts (Suc l))
  if  $L_i$ :  $l \geq i$  and enat: enat (Suc l) < llen Sts for l
proof -
  show  $D \in Q\_of\_state$  (lnth Sts l)
    using  $L_i$  enat
    apply (induction  $l - i$  arbitrary: l)
    subgoal using d by auto
    subgoal using  $d(1)$  in\_Sts\_in\_Sts\_Suc
      by (metis (no\_types, lifting) Suc\_ile\_eq add\_Suc\_right add\_diff\_cancel\_left' le\_SucE
        le\_Suc\_ex less\_imp\_le)
    done
  then show  $D \in Q\_of\_state$  (lnth Sts (Suc l))
    using  $L_i$  enat in\_Sts\_in\_Sts\_Suc by blast
qed
have  $i \leq x \implies \text{enat } x < \text{llen } Sts \implies D \in Q\_of\_state$  (lnth Sts x) for x
  apply (cases x)
  subgoal using  $d(1)$  by (auto intro!: exI[of  $_{-} i$ ] simp: less\_Suc\_eq)
  subgoal for  $x'$ 
    using  $d(1)$   $D\_in\_Sts\_Suc$ [of  $x'$ ] by (cases  $\langle i \leq x' \rangle$ ) (auto simp: not\_less\_eq\_eq)
  done
then have  $D \in \text{Liminf\_llist}$  (lmap  $Q\_of\_state$  Sts)
  unfolding Liminf\_llist\_def by (auto intro!: exI[of  $_{-} i$ ] simp: d)
then show ?thesis
  unfolding Liminf\_state\_def by auto
qed

lemma from_P_to_Q:
  assumes
    fair: fair\_state\_seq Sts and
    ns:  $Ns = \text{lmap}$  grounding\_of\_state Sts and
    c:  $C \in \text{Liminf\_llist}$   $Ns - sr.Rf$  (Liminf\_llist  $Ns$ ) and
    d:  $D \in P\_of\_state$  (lnth Sts i) enat  $i < \text{llen } Sts$  subsumes  $D C$  and
    d\_least:  $\forall E. E \in \{\text{clss\_of\_state } (Sup\_state Sts)\} \wedge \text{subsumes } E C\}. \neg \text{strictly\_subsumes } E D$ 
  shows  $\exists l. D \in Q\_of\_state$  (lnth Sts l)  $\wedge$  enat  $l < \text{llen } Sts$ 
proof -
  let ?Ns =  $\lambda i. N\_of\_state$  (lnth Sts i)
  let ?Ps =  $\lambda i. P\_of\_state$  (lnth Sts i)
  let ?Qs =  $\lambda i. Q\_of\_state$  (lnth Sts i)

  have ground_C: is\_ground\_cls C
    using c using Liminf\_grounding\_of\_state\_ground ns by auto

  have derivns: chain sr\_ext.derive  $Ns$ 
    using resolution\_prover\_ground\_derivation deriv ns by auto

  have  $\exists \sigma. D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$ 
    using instance\_if\_subsumed\_and\_in\_limit ns c d by blast
  then obtain  $\sigma$  where

```

```

σ: D · σ = C is_ground_subst σ
by auto

from deriv have four_ten: chain sr_ext.derive Ns
  using resolution_prover_ground_derivation ns by auto

obtain l where
  Lp: D ∈ P_of_state (lnth Sts l) ∧ D ∉ P_of_state (lnth Sts (Suc l)) ∧ i ≤ l ∧ enat (Suc l) < llength Sts
  using fair using eventually_removed_from_P d unfolding ns by auto
then have LNs: enat (Suc l) < llength Ns
  using ns by auto
from Lp have lnth Sts l ∼ lnth Sts (Suc l)
  using deriv using chain_lnth_rel by auto
then show ?thesis
proof (cases rule: RP.cases)
  case (backward_subsumption_P D' N D_twin P Q)
  note lrhs = this(1,2) and D'_p = this(3,4)
  then have twins: D_twin = D ?Ns (Suc l) = N ?Ns l = N ?Ps (Suc l) = P
    ?Ps l = P ∪ {D_twin} ?Qs (Suc l) = Q ?Qs l = Q
  using Lp by auto
  note D'_p = D'_p[unfolded twins(1)]
  then have subc: subsumes D' C
  unfolding strictly_subsumes_def subsumes_def using σ
  by (metis subst_cls_comp_subst subst_cls_mono_mset)
  from D'_p have D' ∈ class_of_state (Sup_state Sts)
  unfolding twins(2)[symmetric] using Lp
  by (metis (no.types) UnI1 class_of_state_def N_of_state.simps llength_lmap lnth_lmap
    lnth_subset_Sup_llist subsetCE Sup_state_def)
  then have False
  using d_least D'_p subc by auto
  then show ?thesis
  by auto
next
  case (backward_reduction_P E L' N L σ D' P Q)
  then have twins: D' + {#L#} = D ?Ns (Suc l) = N ?Ns l = N ?Ps (Suc l) = P ∪ {D'}
    ?Ps l = P ∪ {D' + {#L#}} ?Qs (Suc l) = Q ?Qs l = Q
  using Lp by auto
  then have D'_p: strictly_subsumes D' D ∧ D' ∈ ?Ps (Suc l)
  using subset_strictly_subsumes[of D' D] by auto
  then have subc: subsumes D' C
  using d(3) subsumes_trans unfolding strictly_subsumes_def by auto
  from D'_p have D' ∈ class_of_state (Sup_state Sts)
  using Lp by (metis (no.types) UnI1 class_of_state_def P_of_state.simps llength_lmap lnth_lmap
    lnth_subset_Sup_llist subsetCE sup_ge2 Sup_state_def)
  then have False
  using d_least D'_p subc by auto
  then show ?thesis
  by auto
next
  case (inference_computation N Q D_twin P)
  then have twins: D_twin = D ?Ps (Suc l) = P ?Ps l = P ∪ {D_twin}
    ?Qs (Suc l) = Q ∪ {D_twin} ?Qs l = Q
  using Lp by auto
  then show ?thesis
  using d σ Lp by auto
qed (use Lp in auto)
qed

lemma variants_sym: variants D D' ⟷ variants D' D
  unfolding variants_def by auto

lemma variants_imp_exists_substitution: variants D D' ⟹ ∃ σ. D · σ = D'
  unfolding variants_iff_subsumes subsumes_def

```

by (meson strictly\_subsumes\_def subset\_mset\_def strict\_subset\_subst\_strictly\_subsumes subsumes\_def)

**lemma** properly\_subsume\_variants:  
**assumes** strictly\_subsumes  $E$   $D$  **and** variants  $D$   $D'$   
**shows** strictly\_subsumes  $E$   $D'$   
**proof** –  
**from** *assms* **obtain**  $\sigma$   $\sigma'$  **where**  
 $\sigma_{\sigma'_p}: D \cdot \sigma = D' \wedge D' \cdot \sigma' = D$   
**using** variants\_imp\_exists\_substitution variants\_sym **by** metis  
  
**from** *assms* **obtain**  $\sigma''$  **where**  
 $E \cdot \sigma'' \subseteq\# D$   
**unfolding** strictly\_subsumes\_def subsumes\_def **by** auto  
**then have**  $E \cdot \sigma'' \cdot \sigma \subseteq\# D \cdot \sigma$   
**using** subst\_cls\_mono\_mset **by** blast  
**then have**  $E \cdot (\sigma'' \odot \sigma) \subseteq\# D'$   
**using**  $\sigma_{\sigma'_p}$  **by** auto  
**moreover from** *assms* **have**  $n: (\nexists \sigma. D \cdot \sigma \subseteq\# E)$   
**unfolding** strictly\_subsumes\_def subsumes\_def **by** auto  
**have**  $\nexists \sigma. D' \cdot \sigma \subseteq\# E$   
**proof**  
**assume**  $\exists \sigma'''. D' \cdot \sigma''' \subseteq\# E$   
**then obtain**  $\sigma'''$  **where**  
 $D' \cdot \sigma''' \subseteq\# E$   
**by** auto  
**then have**  $D \cdot (\sigma \odot \sigma''') \subseteq\# E$   
**using**  $\sigma_{\sigma'_p}$  **by** auto  
**then show** False  
**using**  $n$  **by** metis  
**qed**  
**ultimately show** ?thesis  
**unfolding** strictly\_subsumes\_def subsumes\_def **by** metis  
**qed**

**lemma** neg\_properly\_subsume\_variants:  
**assumes**  $\neg$  strictly\_subsumes  $E$   $D$  **and** variants  $D$   $D'$   
**shows**  $\neg$  strictly\_subsumes  $E$   $D'$   
**using** *assms* properly\_subsume\_variants variants\_sym **by** auto

**lemma** from\_N\_to\_P\_or\_Q:  
**assumes**  
*fair*: fair\_state\_seq *Sts* **and**  
*ns*:  $Ns = \text{lmap } \text{grounding\_of\_state } Sts$  **and**  
*c*:  $C \in \text{Liminf\_l1st } Ns - \text{sr.Rf } (\text{Liminf\_l1st } Ns)$  **and**  
*d*:  $D \in N\_of\_state (\text{lnth } Sts \ i)$  *enat*  $i < \text{length } Sts$  subsumes  $D$   $C$  **and**  
*d.least*:  $\forall E \in \{E. E \in (\text{cls\_of\_state } (\text{Sup\_state } Sts)) \wedge \text{subsumes } E \ C\}. \neg \text{strictly\_subsumes } E \ D$   
**shows**  $\exists l \ D' \ \sigma'. D' \in P\_of\_state (\text{lnth } Sts \ l) \cup Q\_of\_state (\text{lnth } Sts \ l) \wedge$   
 $\text{enat } l < \text{length } Sts \wedge$   
 $(\forall E \in \{E. E \in (\text{cls\_of\_state } (\text{Sup\_state } Sts)) \wedge \text{subsumes } E \ C\}. \neg \text{strictly\_subsumes } E \ D') \wedge$   
 $D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma' \wedge \text{subsumes } D' \ C$   
**proof** –  
**let** ?Ns =  $\lambda i. N\_of\_state (\text{lnth } Sts \ i)$   
**let** ?Ps =  $\lambda i. P\_of\_state (\text{lnth } Sts \ i)$   
**let** ?Qs =  $\lambda i. Q\_of\_state (\text{lnth } Sts \ i)$   
  
**have** *ground\_C*: *is\_ground\_cls*  $C$   
**using** *c* **using** Liminf\_grounding\_of\_state\_ground *ns* **by** auto  
  
**have** *derivns*: *chain sr\_ext.derive* *Ns*  
**using** resolution\_prover\_ground\_derivation *deriv ns* **by** auto  
  
**have**  $\exists \sigma. D \cdot \sigma = C \wedge \text{is\_ground\_subst } \sigma$   
**using** instance\_if\_subsumed\_and\_in\_limit *ns c d* **by** blast

```

then obtain  $\sigma$  where
   $\sigma: D \cdot \sigma = C$  is_ground_subst  $\sigma$ 
  by auto

from  $c$  have no_taut:  $\neg (\exists A. \text{Pos } A \in \# C \wedge \text{Neg } A \in \# C)$ 
  using sr.tautology_redundant by auto

from deriv have four_ten: chain sr_ext.derive  $Ns$ 
  using resolution_prover_ground_derivation ns by auto

have  $\exists l. D \in N\_of\_state (lth Sts l) \wedge D \notin N\_of\_state (lth Sts (Suc l)) \wedge i \leq l \wedge enat (Suc l) < llength Sts$ 
  using fair using eventually_removed_from_N d unfolding ns by auto
then obtain  $l$  where
   $l_p: D \in N\_of\_state (lth Sts l) \wedge D \notin N\_of\_state (lth Sts (Suc l)) \wedge i \leq l \wedge enat (Suc l) < llength Sts$ 
  by auto
then have  $lNs: enat (Suc l) < llength Ns$ 
  using ns by auto
from  $l_p$  have  $lth Sts l \rightsquigarrow lth Sts (Suc l)$ 
  using deriv using chain_lth_rel by auto
then show ?thesis
proof (cases rule: RP.cases)
  case (tautology_deletion A D_twin N P Q)
  then have  $D\_twin = D$ 
    using  $l_p$  by auto
  then have  $\text{Pos } (A \cdot a \sigma) \in \# C \wedge \text{Neg } (A \cdot a \sigma) \in \# C$ 
    using tautology_deletion(3,4)  $\sigma$ 
    by (metis Melem_subst_cls eql_neg_lit_eql_atm eql_pos_lit_eql_atm)
  then have False
    using no_taut by metis
  then show ?thesis
    by blast
next
  case (forward_subsumption D' P Q D_twin N)
  note  $lrhs = this(1,2)$  and  $D'_p = this(3,4)$ 
  then have twins:  $D\_twin = D \text{ ?Ns } (Suc l) = N \text{ ?Ns } l = N \cup \{D\_twin\} \text{ ?Ps } (Suc l) = P$ 
     $\text{ ?Ps } l = P \text{ ?Qs } (Suc l) = Q \text{ ?Qs } l = Q$ 
    using  $l_p$  by auto
  note  $D'_p = D'_p[\text{unfolded twins}(1)]$ 
  from  $D'_p(2)$  have subs: subsumes  $D' C$ 
    using  $d(3)$  by (blast intro: subsumes_trans)
  moreover have  $D' \in \text{cls\_of\_state } (Sup\_state Sts)$ 
    using twins  $D'_p l_p$  unfolding cls_of_state_def Sup_state_def
    by simp (metis (no_types) contra_subsetD llength_lmap lth_lmap lth_subset_Sup_llist)
  ultimately have  $\neg \text{strictly\_subsumes } D' D$ 
    using d_least by auto
  then have subsumes  $D D'$ 
    unfolding strictly_subsumes_def using  $D'_p$  by auto
  then have  $v: \text{variants } D D'$ 
    using  $D'_p$  unfolding variants_iff_subsumes by auto
  then have mini:  $\forall E \in \{E \in \text{cls\_of\_state } (Sup\_state Sts). \text{subsumes } E C\}. \neg \text{strictly\_subsumes } E D'$ 
    using d_least  $D'_p$  neg_properly_subsume_variants[of  $- D D'$ ] by auto

  from  $v$  have  $\exists \sigma'. D' \cdot \sigma' = C$ 
    using  $\sigma$  variants_imp_exists_substitution variants_sym by (metis subst_cls_comp_subst)
  then have  $\exists \sigma'. D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma'$ 
    using ground_C by (meson make_ground_subst refl)
  then obtain  $\sigma'$  where
     $\sigma'_p: D' \cdot \sigma' = C \wedge \text{is\_ground\_subst } \sigma'$ 
    by metis

  show ?thesis
    using  $D'_p$  twins  $l_p$  subs mini  $\sigma'_p$  by auto
next

```

```

case (forward_reduction E L' P Q L σ D' N)
then have twins: D' + {#L#} = D ?Ns (Suc l) = N ∪ {D'} ?Ns l = N ∪ {D' + {#L#}}
  ?Ps (Suc l) = P ?Ps l = P ?Qs (Suc l) = Q ?Qs l = Q
  using L_p by auto
then have D'_p: strictly_subsumes D' D ∧ D' ∈ ?Ns (Suc l)
  using subset_strictly_subsumes[of D' D] by auto
then have subc: subsumes D' C
  using d(3) subsumes_trans unfolding strictly_subsumes_def by blast
from D'_p have D' ∈ clss_of_state (Sup_state Sts)
  using L_p by (metis (no-types) UnI1 clss_of_state_def N_of_state.simps llength_lmap lnth_lmap
    lnth_subset_Sup_llist subsetCE Sup_state_def)
then have False
  using d_least D'_p subc by auto
then show ?thesis
  by auto
next
case (clause_processing N D_twin P Q)
then have twins: D_twin = D ?Ns (Suc l) = N ?Ns l = N ∪ {D} ?Ps (Suc l) = P ∪ {D}
  ?Ps l = P ?Qs (Suc l) = Q ?Qs l = Q
  using L_p by auto
then show ?thesis
  using d σ L_p d_least by blast
qed (use L_p in auto)
qed

lemma eventually_in_Qinf:
assumes
  D_p: D ∈ clss_of_state (Sup_state Sts)
  subsumes D C ∀ E ∈ {E. E ∈ (clss_of_state (Sup_state Sts)) ∧ subsumes E C}. ¬ strictly_subsumes E D and
  fair: fair_state_seq Sts and

  ns: Ns = lmap grounding_of_state Sts and
  c: C ∈ Liminf_llist Ns - sr.Rf (Liminf_llist Ns) and
  ground_C: is_ground_cls C
shows ∃ D' σ'. D' ∈ Q_of_state (Liminf_state Sts) ∧ D' · σ' = C ∧ is_ground_subst σ'
proof -
  let ?Ns = λi. N_of_state (lnth Sts i)
  let ?Ps = λi. P_of_state (lnth Sts i)
  let ?Qs = λi. Q_of_state (lnth Sts i)

  from D_p obtain i where
    i_p: i < llength Sts D ∈ ?Ns i ∨ D ∈ ?Ps i ∨ D ∈ ?Qs i
  unfolding clss_of_state_def Sup_state_def
  by simp_all (metis (no-types) in_Sup_llist_in_nth llength_lmap lnth_lmap)

  have derivns: chain sr_ext.derive Ns using resolution_prover_ground_derivation deriv ns by auto

  have ∃ σ. D · σ = C ∧ is_ground_subst σ
    using instance_if_subsumed_and_in_limit[OF ns c] D_p i_p by blast
  then obtain σ where
    σ: D · σ = C is_ground_subst σ
  by blast

  {
    assume a: D ∈ ?Ns i
    then obtain D' σ' l where D'_p:
      D' ∈ ?Ps l ∪ ?Qs l
      D' · σ' = C
      enat l < llength Sts
      is_ground_subst σ'
      ∀ E ∈ {E. E ∈ (clss_of_state (Sup_state Sts)) ∧ subsumes E C}. ¬ strictly_subsumes E D'
      subsumes D' C
    using from_N_to_P_or_Q deriv fair ns c i_p(1) D_p(2) D_p(3) by blast
  }

```

```

then obtain  $l'$  where
   $l'_p: D' \in ?Qs\ l'\ l' < \text{length } Sts$ 
  using  $\text{from\_P\_to\_Q}[OF\ \text{fair}\ ns\ c - D'_p(3)\ D'_p(6)\ D'_p(5)]$  by blast
then have  $D' \in Q\_of\_state\ (\text{Liminf\_state } Sts)$ 
  using  $\text{from\_Q\_to\_Q\_inf}[OF\ \text{fair}\ ns\ c - l'_p(2)]\ D'_p$  by auto
then have ?thesis
  using  $D'_p$  by auto
}
moreover
{
  assume  $a: D \in ?Ps\ i$ 
  then obtain  $l'$  where
     $l'_p: D \in ?Qs\ l'\ l' < \text{length } Sts$ 
    using  $\text{from\_P\_to\_Q}[OF\ \text{fair}\ ns\ c\ a\ i_p(1)\ D_p(2)\ D_p(3)]$  by auto
  then have  $D \in Q\_of\_state\ (\text{Liminf\_state } Sts)$ 
    using  $\text{from\_Q\_to\_Q\_inf}[OF\ \text{fair}\ ns\ c\ l'_p(1)\ l'_p(2)]\ D_p(3)\ \sigma(1)\ \sigma(2)\ D_p(2)$  by auto
  then have ?thesis
    using  $D_p\ \sigma$  by auto
}
moreover
{
  assume  $a: D \in ?Qs\ i$ 
  then have  $D \in Q\_of\_state\ (\text{Liminf\_state } Sts)$ 
    using  $\text{from\_Q\_to\_Q\_inf}[OF\ \text{fair}\ ns\ c\ a\ i_p(1)]\ \sigma\ D_p(2,3)$  by auto
  then have ?thesis
    using  $D_p\ \sigma$  by auto
}
ultimately show ?thesis
  using  $i_p$  by auto
qed

```

The following corresponds to Lemma 4.11:

**lemma** *fair\_imp\_Liminf\_minus\_Rf\_subset\_ground\_Liminf\_state*:

```

assumes
   $\text{deriv}: \text{chain } (op \rightsquigarrow) Sts$  and
   $\text{fair}: \text{fair\_state\_seq } Sts$  and
   $ns: Ns = \text{lmap } \text{grounding\_of\_state } Sts$ 
shows  $\text{Liminf\_llist } Ns - sr.Rf\ (\text{Liminf\_llist } Ns) \subseteq \text{grounding\_of\_cls } (Q\_of\_state\ (\text{Liminf\_state } Sts))$ 
proof
  let  $?Ns = \lambda i. N\_of\_state\ (\text{lnth } Sts\ i)$ 
  let  $?Ps = \lambda i. P\_of\_state\ (\text{lnth } Sts\ i)$ 
  let  $?Qs = \lambda i. Q\_of\_state\ (\text{lnth } Sts\ i)$ 

```

```

have  $SQinf: \text{cls\_of\_state } (\text{Liminf\_state } Sts) = \text{Liminf\_llist } (\text{lmap } Q\_of\_state\ Sts)$ 
  using fair unfolding fair\_state\_seq\_def Liminf\_state\_def cls\_of\_state\_def by auto

```

**fix**  $C$

**assume**  $C_p: C \in \text{Liminf\_llist } Ns - sr.Rf\ (\text{Liminf\_llist } Ns)$

**then have**  $C \in \text{Sup\_llist } Ns$

**using**  $\text{Liminf\_llist\_subset\_Sup\_llist}[of\ Ns]$  **by** *blast*

**then obtain**  $D\_proto$  **where**

$D\_proto \in \text{cls\_of\_state } (\text{Sup\_state } Sts) \wedge \text{subsumes } D\_proto\ C$

**using**  $\text{in\_Sup\_llist\_in\_Sup\_state}$  **unfolding**  $ns$  *subsumes\\_def* **by** *blast*

**then obtain**  $D$  **where**

$D_p: D \in \text{cls\_of\_state } (\text{Sup\_state } Sts)$

$\text{subsumes } D\ C$

$\forall E \in \{E. E \in \text{cls\_of\_state } (\text{Sup\_state } Sts) \wedge \text{subsumes } E\ C\}. \neg \text{strictly\_subsumes } E\ D$

**using**  $\text{strictly\_subsumes\_has\_minimum}[of\ \{E. E \in \text{cls\_of\_state } (\text{Sup\_state } Sts) \wedge \text{subsumes } E\ C\}]$

**by** *auto*

**have**  $\text{ground\_C}: \text{is\_ground\_cls } C$

**using**  $C_p$  **using**  $\text{Liminf\_grounding\_of\_state\_ground } ns$  **by** *auto*

```

have  $\exists D' \sigma'. D' \in Q\_of\_state (Liminf\_state\ Sts) \wedge D' \cdot \sigma' = C \wedge is\_ground\_subst\ \sigma'$ 
  using eventually_in_Qinf[of D C Ns] using D_p(1) D_p(2) D_p(3) fair ns C_p ground_C by auto
then obtain  $D' \sigma'$  where
   $D'_p: D' \in Q\_of\_state (Liminf\_state\ Sts) \wedge D' \cdot \sigma' = C \wedge is\_ground\_subst\ \sigma'$ 
  by blast
then have  $D' \in clss\_of\_state (Liminf\_state\ Sts)$ 
  by (simp add: clss_of_state_def)
then have  $C \in grounding\_of\_state (Liminf\_state\ Sts)$ 
  unfolding grounding_of_clss_def grounding_of_cls_def using  $D'_p$  by auto
then show  $C \in grounding\_of\_clss (Q\_of\_state (Liminf\_state\ Sts))$ 
  using SQinf clss_of_state_def fair fair_state_seq_def by auto
qed

```

The following corresponds to (one direction of) Theorem 4.13:

**lemma** *ground\_subclauses*:

```

assumes
   $\forall i < length\ CAs. CAs\ !\ i = Cs\ !\ i + poss\ (AAs\ !\ i)$  and
   $length\ Cs = length\ CAs$  and
   $is\_ground\_cls\_list\ CAs$ 
shows  $is\_ground\_cls\_list\ Cs$ 
  unfolding is_ground_cls_list_def
  by (metis assms in_set_conv_nth is_ground_cls_list_def is_ground_cls_union)

```

**lemma** *subseq\_Liminf\_state\_eventually\_always*:

```

fixes CC
assumes
  finite CC and
   $CC \neq \{\}$  and
   $CC \subseteq Q\_of\_state (Liminf\_state\ Sts)$ 
shows  $\exists j. enat\ j < llength\ Sts \wedge (\forall j' \geq enat\ j. j' < llength\ Sts \longrightarrow CC \subseteq Q\_of\_state (lnth\ Sts\ j'))$ 
proof -
  from assms(3) have  $\forall C \in CC. \exists j. enat\ j < llength\ Sts \wedge$ 
     $(\forall j' \geq enat\ j. j' < llength\ Sts \longrightarrow C \in Q\_of\_state (lnth\ Sts\ j'))$ 
    unfolding Liminf_state_def Liminf_llist_def by force
  then obtain f where
     $f\_p: \forall C \in CC. f\ C < llength\ Sts \wedge (\forall j' \geq enat\ (f\ C). j' < llength\ Sts \longrightarrow C \in Q\_of\_state (lnth\ Sts\ j'))$ 
    by moura

```

**define**  $j :: nat$  **where**

$j = Max\ (f\ ` CC)$

**have**  $enat\ j < llength\ Sts$

**unfolding**  $j\_def$  **using**  $f\_p\ assms(1)$

**by** (metis (mono\_tags) Max.in assms(2) finite\_imageI imageE image\_is\_empty)

**moreover have**  $\forall C\ j'. C \in CC \longrightarrow enat\ j \leq j' \longrightarrow j' < llength\ Sts \longrightarrow C \in Q\_of\_state (lnth\ Sts\ j')$

**proof** (intro allI impI)

**fix**  $C :: 'a\ clause$  **and**  $j' :: nat$

**assume**  $a: C \in CC\ enat\ j \leq enat\ j'\ enat\ j' < llength\ Sts$

**then have**  $f\ C \leq j'$

**unfolding**  $j\_def$  **using**  $assms(1)\ Max.bounded\_iff$  **by** auto

**then show**  $C \in Q\_of\_state (lnth\ Sts\ j')$

**using**  $f\_p\ a$  **by** auto

**qed**

**ultimately show** *?thesis*

**by** auto

**qed**

**lemma** *empty\_clause\_in\_Q\_of\_Liminf\_state*:

**assumes**

$empty\_in: \{\#\} \in Liminf\_llist (lmap\ grounding\_of\_state\ Sts)$  **and**

$fair: fair\_state\_seq\ Sts$

**shows**  $\{\#\} \in Q\_of\_state (Liminf\_state\ Sts)$

**proof** -



```

define Ns :: 'a clause set llist where
  ns: Ns = lmap grounding_of_state Sts

from empty_in have in_Liminf_not_Rf:  $\{\#\} \in \text{Liminf\_llist } Ns - sr.Rf (\text{Liminf\_llist } Ns)$ 
  unfolding ns sr.Rf_def by auto

from assms obtain i where
  i_p: enat i < llength (lmap grounding_of_state Sts)
   $\{\#\} \in \text{lnth } (\text{lmap } \text{grounding\_of\_state } Sts) \ i$ 
  unfolding Liminf_llist_def by force
then have  $\{\#\} \in \text{grounding\_of\_state } (\text{lnth } Sts \ i)$ 
  by auto
then have  $\{\#\} \in \text{clss\_of\_state } (\text{lnth } Sts \ i)$ 
  unfolding grounding_of_clss_def grounding_of_cls_def by auto
then have in_Sup_state:  $\{\#\} \in \text{clss\_of\_state } (\text{Sup\_state } Sts)$ 
  using i_p(1) unfolding Sup_state_def clss_of_state_def
  by simp (metis llength_lmap lnth_lmap lnth_subset_Sup_llist set_mp)
then have  $\exists D' \sigma'. D' \in Q\_of\_state (\text{Liminf\_state } Sts) \wedge D' \cdot \sigma' = \{\#\} \wedge \text{is\_ground\_subst } \sigma'$ 
  using eventually_in_Qinf[of  $\{\#\} \ \{\#\}$  Ns, OF in_Sup_state  $\_ \_ \text{fair } ns \text{ in\_Liminf\_not\_Rf}$ ]
  unfolding is_ground_cls_def strictly_subsumes_def subsumes_def by simp
then show ?thesis
  by simp
qed

lemma grounding_of_state_Liminf_state_subseteq:
  grounding_of_state (Liminf_state Sts)  $\subseteq \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_state } Sts)$ 
proof
  fix C :: 'a clause
  assume C  $\in \text{grounding\_of\_state } (\text{Liminf\_state } Sts)$ 
  then obtain D  $\sigma$  where
    D $\sigma$  $\_p$ : D  $\in \text{clss\_of\_state } (\text{Liminf\_state } Sts)$  D  $\cdot \sigma = C$  is\_ground\_subst  $\sigma$ 
    unfolding clss_of_state_def grounding_of_clss_def grounding_of_cls_def by auto
  then have ii: D  $\in \text{Liminf\_llist } (\text{lmap } N\_of\_state \ Sts) \vee$ 
    D  $\in \text{Liminf\_llist } (\text{lmap } P\_of\_state \ Sts) \vee$ 
    D  $\in \text{Liminf\_llist } (\text{lmap } Q\_of\_state \ Sts)$ 
    unfolding clss_of_state_def Liminf_state_def by simp
  then have C  $\in \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_clss } (\text{lmap } N\_of\_state \ Sts)) \vee$ 
    C  $\in \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_clss } (\text{lmap } P\_of\_state \ Sts)) \vee$ 
    C  $\in \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_clss } (\text{lmap } Q\_of\_state \ Sts))$ 
    unfolding Liminf_llist_def grounding_of_clss_def grounding_of_cls_def
    apply  $\_$ 
    apply (erule disjE)
    subgoal
      apply (rule disjI1)
      using D $\sigma$  $\_p$  by auto
    subgoal
      apply (erule HOL.disjE)
      subgoal
        apply (rule disjI2)
        apply (rule disjI1)
        using D $\sigma$  $\_p$  by auto
      subgoal
        apply (rule disjI2)
        apply (rule disjI2)
        using D $\sigma$  $\_p$  by auto
      done
    done
  then show C  $\in \text{Liminf\_llist } (\text{lmap } \text{grounding\_of\_state } Sts)$ 
    unfolding Liminf_llist_def clss_of_state_def grounding_of_clss_def by auto
qed

theorem RP_sound:
  assumes  $\{\#\} \in \text{clss\_of\_state } (\text{Liminf\_state } Sts)$ 

```

```

shows  $\neg$  satisfiable (grounding_of_state (lhd Sts))
proof -
  from assms have  $\{\#\} \in$  grounding_of_state (Liminf_state Sts)
    unfolding grounding_of_cls_def by (force intro: ex_ground_subst)
  then have  $\neg$  satisfiable (grounding_of_state (Liminf_state Sts))
    unfolding true_cls_def by auto
  then have  $\neg$  satisfiable (Liminf_llist (lmap grounding_of_state Sts))
    using grounding_of_state_Liminf_state_subseteq true_cls_mono by blast
  then have  $\neg$  satisfiable (lhd (lmap grounding_of_state Sts))
    using sr_ext.sat_deriv_Liminf_iff[of lmap grounding_of_state Sts]
    by (metis deriv_resolution_prover_ground_derivation)
  then show ?thesis
    unfolding lhd_lmap_Sts .
qed

theorem RP_saturated_if_fair:
  assumes fair: fair_state_seq Sts
  shows sr.saturated_upto (Liminf_llist (lmap grounding_of_state Sts))
proof -
  define Ns :: 'a clause set llist where
    ns: Ns = lmap grounding_of_state Sts

  let ?N =  $\lambda i.$  grounding_of_state (lnth Sts i)

  let ?Ns =  $\lambda i.$  N_of_state (lnth Sts i)
  let ?Ps =  $\lambda i.$  P_of_state (lnth Sts i)
  let ?Qs =  $\lambda i.$  Q_of_state (lnth Sts i)

  have ground_ns_in_ground_limit_st:
    Liminf_llist Ns - sr.Rf (Liminf_llist Ns)  $\subseteq$  grounding_of_cls (Q_of_state (Liminf_state Sts))
    using fair deriv fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state ns by blast

  have derivns: chain sr_ext.derive Ns
    using resolution_prover_ground_derivation deriv ns by auto

  {
    fix  $\gamma ::$  'a inference
    assume  $\gamma_p$ :  $\gamma \in$  gr.ord_Γ
    let ?CC = side_prem_of  $\gamma$ 
    let ?DA = main_prem_of  $\gamma$ 
    let ?E = concl_of  $\gamma$ 
    assume a: set_mset ?CC  $\cup$  { ?DA }
       $\subseteq$  Liminf_llist (lmap grounding_of_state Sts) - sr.Rf (Liminf_llist (lmap grounding_of_state Sts))

    have ground_ground_Liminf: is_ground_cls (Liminf_llist (lmap grounding_of_state Sts))
      using Liminf_grounding_of_state_ground unfolding is_ground_cls_def by auto

    have ground_cc: is_ground_cls (set_mset ?CC)
      using a ground_ground_Liminf is_ground_cls_def by auto

    have ground_da: is_ground_cls ?DA
      using a grounding_ground_singletonI ground_ground_Liminf
      by (simp add: Liminf_grounding_of_state_ground)

    from  $\gamma_p$  obtain CAs AAs As where
      CAs_p: gr.ord_resolve CAs ?DA AAs As ?E  $\wedge$  mset CAs = ?CC
      unfolding gr.ord_Γ_def by auto

    have DA_CAs_in_ground_Liminf:
      { ?DA }  $\cup$  set CAs  $\subseteq$  grounding_of_cls (Q_of_state (Liminf_state Sts))
      using a CAs_p unfolding cls_of_state_def using fair unfolding fair_state_seq_def
      by (metis (no_types, lifting) Un_empty_left ground_ns_in_ground_limit_st a cls_of_state_def
        ns set_mset_mset subset_trans sup_commute)
  }

```

```

then have ground_cas: is_ground_cls_list CAs
  using CAs_p unfolding is_ground_cls_list_def by auto

have ground_e: is_ground_cls ?E
proof –
  have a1: atms_of ?E  $\subseteq$  ( $\bigcup$  CA  $\in$  set CAs. atms_of CA)  $\cup$  atms_of ?DA
    using  $\gamma$ -p ground_cc ground_da gr.ord.resolve_atms_of.concl_subset[of CAs ?DA _ _ ?E] CAs_p
    by auto
  {
    fix L :: 'a literal
    assume L  $\in$  # concl_of  $\gamma$ 
    then have atm_of L  $\in$  atms_of (concl_of  $\gamma$ )
      by (meson atm_of_lit.in_atms_of)
    then have is_ground_atm (atm_of L)
      using a1 ground_cas ground_da is_ground_cls_imp_is_ground_atm is_ground_cls_list_def
      by auto
  }
  then show ?thesis
    unfolding is_ground_cls_def is_ground_lit_def by simp
qed

have  $\exists$  AAAs As  $\sigma$ . ord_resolve (S_M S (Q_of_state (Liminf_state Sts))) CAs ?DA AAAs As  $\sigma$  ?E
  using CAs_p[THEN conjunct1]
proof (cases rule: gr.ord_resolve.cases)
  case (ord_resolve n Cs D)
    note DA = this(1) and e = this(2) and cas_len = this(3) and cs_len = this(4) and
      aas_len = this(5) and as_len = this(6) and nz = this(7) and cas = this(8) and
      aas_not_empty = this(9) and as_aas = this(10) and eligibility = this(11) and
      str_max = this(12) and sel_empty = this(13)

    have len_aas_len_as: length AAAs = length As
      using aas_len as_len by auto

    from as_aas have  $\forall i < n$ .  $\forall A \in \#$  add_mset (As ! i) (AAAs ! i). A = As ! i
      using ord_resolve by simp
    then have  $\forall i < n$ . card (set_mset (add_mset (As ! i) (AAAs ! i)))  $\leq$  Suc 0
      using all_the_same by metis
    then have  $\forall i < \text{length } AAAs$ . card (set_mset (add_mset (As ! i) (AAAs ! i)))  $\leq$  Suc 0
      using aas_len by auto
    then have  $\forall AA \in \text{set } (\text{map2 } \text{add\_mset } As \ AAAs)$ . card (set_mset AA)  $\leq$  Suc 0
      using set_map2_ex[of AAAs As add_mset, OF len_aas_len_as] by auto
    then have is_unifiers id_subst (set_mset ' set (map2 add_mset As AAAs))
      unfolding is_unifiers_def is_unifier_def by auto
    moreover have finite (set_mset ' set (map2 add_mset As AAAs))
      by auto
    moreover have  $\forall AA \in \text{set\_mset ' set } (\text{map2 } \text{add\_mset } As \ AAAs)$ . finite AA
      by auto
    ultimately obtain  $\sigma$  where
       $\sigma$ -p: Some  $\sigma$  = mgu (set_mset ' set (map2 add_mset As AAAs))
      using mgu_complete by metis

    have ground_elig: gr.eligible As (D + negs (mset As))
      using ord_resolve by simp
    have ground_cs:  $\forall i < n$ . is_ground_cls (Cs ! i)
      using ord_resolve(8) ord_resolve(3,4) ground_cas
      using ground_subclauses[of CAs Cs AAAs] unfolding is_ground_cls_list_def by auto
    have ground_set_as: is_ground_atms (set As)
      using ord_resolve(1) ground_da
      by (metis atms_of_negs is_ground_cls_union set_mset_mset is_ground_cls_is_ground_atms.atms_of)
    then have ground_mset_as: is_ground_atm_mset (mset As)
      unfolding is_ground_atm_mset_def is_ground_atms_def by auto
    have ground_as: is_ground_atm_list As

```

```

    using ground_set_as is_ground_atm_list_def is_ground_atms_def by auto
  have ground_d: is_ground_cls D
    using ground_da ord_resolve by simp

  from as.len nz have atms_of D  $\cup$  set As  $\neq \{\}$  finite (atms_of D  $\cup$  set As)
    by auto
  then have Max (atms_of D  $\cup$  set As)  $\in$  atms_of D  $\cup$  set As
    using Max.in by metis
  then have is_ground_Max: is_ground_atm (Max (atms_of D  $\cup$  set As))
    using ground_d ground_mset_as is_ground_cls_imp_is_ground_atm
    unfolding is_ground_atm_mset_def by auto
  then have Max $\sigma$ _is_Max:  $\forall \sigma. \text{Max (atms\_of D } \cup \text{ set As)} \cdot a \sigma = \text{Max (atms\_of D } \cup \text{ set As)}$ 
    by auto

  have ann1: maximal_wrt (Max (atms_of D  $\cup$  set As)) (D + negs (mset As))
    unfolding maximal_wrt_def
    by clarsimp (metis Max.less_iff UnCI  $\langle$ atms_of D  $\cup$  set As  $\neq \{\}$  $\rangle$ 
       $\langle$ finite (atms_of D  $\cup$  set As) $\rangle$  ground_d ground_set_as infinite_growing is_ground_Max
      is_ground_atms_def is_ground_cls_imp_is_ground_atm less_atm_ground)

  from ground_elig have ann2:
    Max (atms_of D  $\cup$  set As)  $\cdot a \sigma = \text{Max (atms\_of D } \cup \text{ set As)}$ 
    D  $\cdot \sigma$  + negs (mset As  $\cdot$  am  $\sigma$ ) = D + negs (mset As)
    using is_ground_Max ground_mset_as ground_d by auto

  from ground_elig have fo_elig:
    eligible (S_M S (Q_of_state (Liminf_state Sts)))  $\sigma$  As (D + negs (mset As))
    unfolding gr.eligible.simps eligible.simps gr.maximal_wrt_def using ann1 ann2
    by (auto simp: S_Q_def)

  have l:  $\forall i < n. \text{gr.strictly\_maximal\_wrt (As ! } i) (Cs ! i)$ 
    using ord_resolve by simp
  then have  $\forall i < n. \text{strictly\_maximal\_wrt (As ! } i) (Cs ! i)$ 
    unfolding gr.strictly_maximal_wrt_def strictly_maximal_wrt_def
    using ground_as[unfolded is_ground_atm_list_def] ground_cs as.len less_atm_ground
    by clarsimp (fastforce simp: is_ground_cls_as_atms)+

  then have ll:  $\forall i < n. \text{strictly\_maximal\_wrt (As ! } i \cdot a \sigma) (Cs ! i \cdot \sigma)$ 
    by (simp add: ground_as ground_cs as.len)

  have m:  $\forall i < n. S_Q (CAs ! i) = \{\#\}$ 
    using ord_resolve by simp

  have ground_e: is_ground_cls ( $\bigcup \# \text{mset Cs} + D$ )
    using ground_d ground_cs ground_e e by simp
  show ?thesis
    using ord_resolve.intros[OF cas.len cs.len aas.len as.len nz cas aas.not_empty  $\sigma$ _p fo_elig ll] m DA e ground_e
    unfolding S_Q_def by auto
qed
then obtain AAs As  $\sigma$  where
   $\sigma$ _p: ord_resolve (S_M S (Q_of_state (Liminf_state Sts))) CAs ?DA AAs As  $\sigma$  ?E
  by auto
then obtain  $\eta s' \eta' \eta 2' CAs' DA' AAs' As' \tau' E'$  where  $s$ _p:
  is_ground_subst  $\eta'$ 
  is_ground_subst_list  $\eta s'$ 
  is_ground_subst  $\eta 2'$ 
  ord_resolve_rename S CAs' DA' AAs' As'  $\tau' E'$ 
  CAs'  $\cdot$  cl  $\eta s' = CAs$ 
  DA'  $\cdot \eta' = ?DA$ 
  E'  $\cdot \eta 2' = ?E$ 
  {DA'}  $\cup$  set CAs'  $\subseteq$  Q_of_state (Liminf_state Sts)
  using ord_resolve_rename_lifting[OF sel_stable, of Q_of_state (Liminf_state Sts) CAs ?DA]
   $\sigma$ _p selection_axioms DA_CAs_in_ground_Liminf by metis

```

```

from this(8) have  $\exists j. \text{enat } j < \text{length } \text{Sts} \wedge (\text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ j)$ 
  unfolding Liminf_llist_def
  using subseteq_Liminf_state_eventually_always[of  $\{\text{DA}'\} \cup \text{set } \text{CAs}'$ ] by auto
then obtain j where
  j-p: is_least ( $\lambda j. \text{enat } j < \text{length } \text{Sts} \wedge \text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ j$ ) j
  using least_exists[of  $\lambda j. \text{enat } j < \text{length } \text{Sts} \wedge \text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ j$ ] by force
then have j-p':  $\text{enat } j < \text{length } \text{Sts} \wedge \text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ j$ 
  unfolding is_least_def by auto
then have jn0:  $j \neq 0$ 
  using empty_Q0 by (metis bot_eq_sup_iff gr_implies_not_zero insert_not_empty length_null
    lnth_0_conv.lhd sup.orderE)
then have j_adds_CAs':  $\neg \text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ (j - 1) \wedge \text{set } \text{CAs}' \cup \{\text{DA}'\} \subseteq ?Qs \ j$ 
  using j-p unfolding is_least_def
  apply (metis (no_types) One_nat_def Suc_diff_Suc Suc_ile_eq diff_diff_cancel diff_zero
    less_imp_le less_one neq0_conv zero_less_diff)
  using j-p'(2) by blast
have lnth Sts (j - 1)  $\rightsquigarrow$  lnth Sts j
  using j-p'(1) jn0 deriv chain_lnth_rel[of - j - 1] by force
then obtain C' where C'-p:
  ?Ns (j - 1) = {}
  ?Ps (j - 1) = ?Ps j  $\cup$  {C'}
  ?Qs j = ?Qs (j - 1)  $\cup$  {C'}
  ?Ns j = concls_of (ord_FO_resolution.inferences_between (?Qs (j - 1)) C')
  C'  $\in$  set CAs'  $\cup$  {DA'}
  C'  $\notin$  ?Qs (j - 1)
  using j_adds_CAs' by (induction rule: RP.cases) auto

then have ihih:  $\text{set } \text{CAs}' \cup \{\text{DA}'\} - \{C'\} \subseteq ?Qs \ (j - 1)$ 
  using j_adds_CAs' by auto
have E'  $\in$  ?Ns j
proof -
  have E'  $\in$  concls_of (ord_FO_resolution.inferences_between (Q_of_state (lnth Sts (j - 1))) C')
  unfolding infer_from_def ord_FO_def unfolding inference_system.inferences_between_def
  apply (rule_tac x = Infer (mset CAs') DA' E' in image_eqI)
  subgoal by auto
  subgoal
    using s-p(4)
    unfolding infer_from_def
    apply (rule ord_resolve_rename.cases)
    using s-p(4)
    using C'-p(3) C'-p(5) j-p'(2) apply force
  done
done
then show ?thesis
  using C'-p(4) by auto
qed
then have E'  $\in$  clss_of_state (lnth Sts j)
  using j-p' unfolding clss_of_state_def by auto
then have ?E  $\in$  grounding_of_state (lnth Sts j)
  using s-p(7) s-p(3) unfolding grounding_of_clss_def grounding_of_cls_def by force
then have  $\gamma \in \text{sr.Ri}$  (grounding_of_state (lnth Sts j))
  using sr.Ri_effective  $\gamma$ -p by auto
then have  $\gamma \in \text{sr.ext.Ri}$  (?N j)
  unfolding sr_ext_Ri_def by auto
then have  $\gamma \in \text{sr.ext.Ri}$  (Sup_llist (lmap grounding_of_state Sts))
  using j-p' contra_subsetD length_lmap lnth_lmap lnth_subset_Sup_llist sr_ext.Ri_mono by metis
then have  $\gamma \in \text{sr.ext.Ri}$  (Liminf_llist (lmap grounding_of_state Sts))
  using sr_ext.Ri_Sup_subset_Ri_Liminf[of Ns] derivns ns by blast
}
then have sr_ext.saturated_upto (Liminf_llist (lmap grounding_of_state Sts))
  unfolding sr_ext.saturated_upto_def sr_ext.inferences_from_def infer_from_def sr_ext_Ri_def
  by auto
then show ?thesis

```

```

    using gd_ord.Γ_ngd_ord.Γ sr.redundancy_criterion_axioms
      redundancy_criterion_standard_extension_saturated_upto_iff[of gr_ord.Γ]
    unfolding sr_ext.Ri_def by auto
qed

corollary RP_complete_if_fair:
  assumes
    fair: fair_state_seq Sts and
    unsat: ¬ satisfiable (grounding_of_state (lhd Sts))
  shows {#} ∈ Q_of_state (Liminf_state Sts)
proof -
  have ¬ satisfiable (Liminf_llist (lmap grounding_of_state Sts))
  unfolding sr_ext.sat_deriv_Liminf_iff[OF resolution_prover_ground_derivation[OF deriv]]
  by (rule unsat[folded lhd_lmap_Sts[of grounding_of_state]])
  moreover have sr.saturated_upto (Liminf_llist (lmap grounding_of_state Sts))
  by (rule RP_saturated_if_fair[OF fair, simplified])
  ultimately have {#} ∈ Liminf_llist (lmap grounding_of_state Sts)
  using sr.saturated_upto_complete_if by auto
  then show ?thesis
  using empty_clause_in_Q_of_Liminf_state fair by auto
qed

end

end

end

```